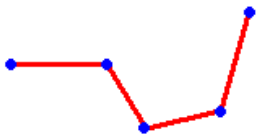
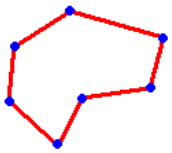
# 1. ÜÇGENLEŞTİRME

Basit geometrik şekilleri ifade etmek için kullanılan en ilkel geometrik nesneler nokta, doğru, üçgen ve poligonlardır.

Tezimizde kullanacağımız temel geometrik kavramlar köşe, kenar, üçgen ve poligondur. Köşe, nokta ile aynı anlama gelmektedir. Kenar ise iki köşe noktasını birleştiren doğru parçasıdır ve kenar iki köşe noktası ile ifade edilir. Birden fazla kenardan oluşan geometrik nesneye ise çok kenarlı denir (Bkz. Şekil 1.1.). Çok kenarlıda n adet kenar için $(n+1)$ adet köşe noktası mevcuttur. Çok kenarlının ilk köşe ile son köşe kenar çizgisi ile birleştirilirse oluşacak nesneye poligon denir. Poligonda ise n adet kenar için n adet köşe noktası mevcuttur. Poligon, noktaların birleştirilmesi sonucunda ortaya çıkan kapalı yüzeylere verilen addır (Bkz. Şekil 1.2.). Poligonun komşu olmayan kenarlar birbirlerini kesmiyor ise poligona basit poligon denir. Tezimizde poligonlar olarak basit poligonlar kullanılacaktır.



Şekil 1.1. Çok kenarlı



Şekil 1.2. Poligon

Noktayı ifade etmek için iki boyutlu düzlemde $(x_i, y_i)$ ifadesi, üç boyutlu düzlemde ise $(x_i, y_i, z_i)$ ifadesi kullanılacaktır. Noktalar kümesini ise $P=\{p_i=(x_i, y_i),\ x_i, y_i \in R \text{ ve } i=1,2,..,N\}$ ile göstereceğiz. Bütün P noktalarını içine alan bölgeyi ifade etmek için bölge ifadesi kullanılacak olup bölge D ile gösterilecektir. D bölgesi kapalı çokgendir. D'nin sınır çizgileri birbirini kesmez. Genel olarak D bölgesi P noktalarının dış bükey bileşenidir, fakat D'nin her zaman dış bükey olması da gerekmemektedir.

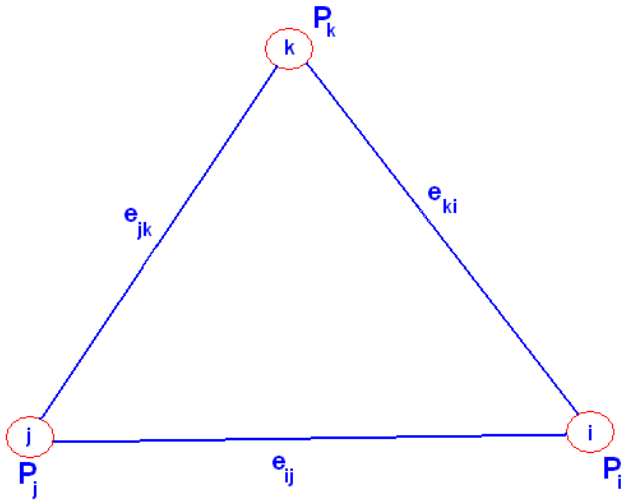
Eğer, bir D bölgesinde alınan herhangi iki nokta bir eğri ile birleştirile-biliniyorsa D bölgesi bağlantılıdır denir.

Bir D bölgesinin üçgenlere bölünmesine üçgenleştirme denir ve üçgenlerin köşe noktalarını P kümesinin noktaları ile ifade ederiz.

Bir bölgenin üçgenlere bölünmesiyle üçgenlerin bütün belirsizlikleri ortadan kaldıracağı beklenmemelidir. Amaç, üçgenler yardımıyla bölge üzerindeki işlemleri daha kolay ve anlaşılır yapmaktır.

Artık, D bölgesinde üçgenler topluluğunu kullanacağımıza için üçgenlerle alakalı bazı formül, gösterim ve kriterleri belirteceğiz.

$P_i$, $P_j$ ve $P_k$ ($P_i=(x_i,y_i)$ noktası ) köşelerinden oluşan basit üçgeni bundan sonra "$T_{ijk}$" şeklinde ifade edeceğiz (Bkz. Şekil 1.3.). Yani, bir üçgeni üç köşe noktası ile ifade edeceğiz ve noktalar saat yönü sırası ile damgalanacaktır. Üç noktanın oluşturduğu küme ise "$I_t$" ile gösterilecektir. $T_{ijk}$ üçgeninin iç bölgesini "$Int(T_{ijk})$" ile göstereceğiz. İki boyutlu düzlemde "$T_{ijk}$" kapalı, "$Int(T_{ijk})$" ise açıktır. $P_i$ ve $P_j$ köşelerini birleştiren çizginin gösterimi ise "$e_{ij}$" kenarı şeklinde olacaktır.




Şekil 1.3. Üçgen

### 1.1. Üçgenleştirme Şartları

i. Her i,j,k $\in$ $I_t$ ve her $T_{ijk}$ üçgeni için $p_i$, $p_j$, $p_k$ köşeleri aynı doğrultu üzerinde değildir.

ii. Her hangi iki üçgenin iç bölgelerinin kesişimi boş kümedir. Yani, i,j,k $\in$ $I_t$ ve α,β,γ $\in$ $I_t$ ise $Int(T_{ijk}) \cap Int(T_{\alpha\beta\gamma}) = \varnothing$ dir.

iii. Her hangi iki üçgenin sınır bölgesinin kesişimi sadece bir kenar çizgisi olabilir.

iv. Bütün üçgenlerin bileşimi D bölgesini meydana getirir. Yani, $D = \bigcup_{ijk \in I_t} T_{ijk}$

Üçgenleştirme algoritmalarının daha iyi anlaşılabilmesi için önce poligonları ve dış bükey kabuk kavramını inceleyelim.

### 1.2. Poligonlar

Düzlemde bulunan n adet noktanın oluşturduğu küme $P=\{p_0, p_1, .. , p_{n-1}\}$ olsun. Noktaların damga sırası dairesel sıralı olsun, yani $p_{n-1}$ noktasından sonra $p_0$ gelsin [yani, $(n-1)+1 \equiv n \equiv 0 (\mathrm{mod}\, n)$ olsun]. Ayrıca, $e_0=p_0p_1$, $e_1=p_1p_2$, .. , $e_{n-1}=p_{n-1}p_0$ noktaları birbirine bağlayan n adet doğru parçası olsun. Eğer

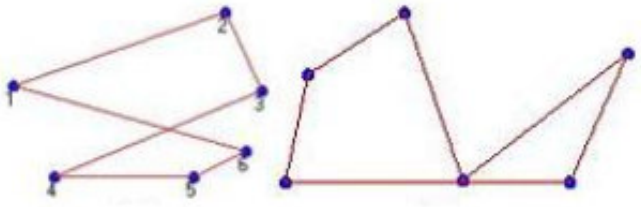
i) $e_i \cap e_j = p_j$ , j = I+1 (mod n) $\forall i, j = 0,..,n-1$

ii) $e_i \cap e_j = \phi$ , j ≠ I+1 (mod n)

şartları sağlanıyor ise $P=\{p_0, p_1, .. , p_{n-1}\}$ noktalar kümesi bir poligon tanımlar (O'Rourke, Sf:1).

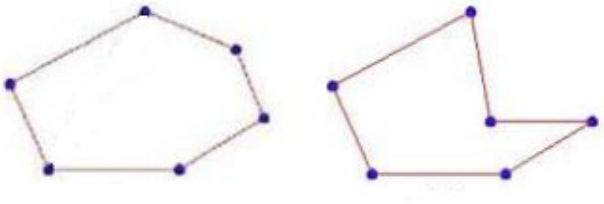
Açıkça poligonlar kapalı bölgelerdir.

Eğer her hangi iki kenar bir birini köşe noktaları haricinde bir noktada kesiyorsa bu şekil basit poligon değildir (Bkz. Şekil 1.4.).



Şekil 1.4. Basit poligon değil

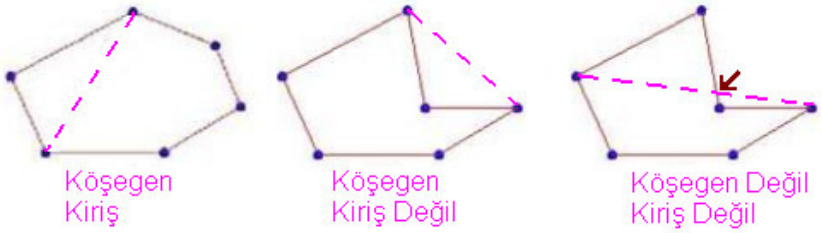
Bir poligonda bütün köşelerinin iç açıları $180^0$ den küçük ise bu poligona dış bükey poligon denir. En az bir köşenin iç açısı $180^0$ den büyük poligona ise iç bükey poligon denir (Bkz. Şekil 1.5.).



Şekil 1.5. Dış bükey, iç bükey

Poligonun köşe damga numaraları saat yönünün tersi yönde sıralı olsun. Poligonun sınırını ∂P ile gösterilir. Poligonun sınır çizgileri üzerinde saat yönünün tersi yönde bütün noktalara da uğrayarak yürüdüğümüzü varsayarsak, bu yürüyüşte poligonun iç bölgesinin hep sol tarafımızda kaldığı gözden kaçırılmamalıdır (O'Rourke, sf:2).

Komşu olmayan iki köşeyi birleştiren ve kenar doğrularını kesmeyen doğru parçasına köşegen denir. Köşegenin tamamı poligonun iç bölgesinde kalıyor ise köşegene kiriş denir (Bkz. Şekil 1.6.). Tezimizde kullanılacak köşegen ifadesi ile kiriş kastedilecektir.




Şekil 1.6. Köşegen, kiriş

Şimdi, doğruların düzlemde kesişimleri ile ilgili bazı ön teoremleri inceleyelim.

**Ön teorem:** Düzlemde üçü aynı doğrultuda olmayan n adet noktadan geçen doğru sayısı en çok `C(n,2)` sayısı ile verilir. Burada,

$$C(n,2)=\frac{(n)!}{C(n-2)!C(2)!}=\frac{(n)(n-1)C(n-2)!}{C(n-2)!(2)}=\frac{(n)(n-1)}{2}$$ dır.

**Kanıt :** Düzlemdeki n noktadan biri sabit kabul edilip geri kalan (n – 1) nokta ile birleştirilmesi ile (n – 1) adet doğru elde edilebilir. Kalan (n – 1) nokta için aynı işlem uygulanıp yine bir nokta sabit tutularak geri kalan (n – 2) nokta ile birleştirilmesinden (n – 2) adet doğru elde edilir. İşlem bu şekilde devam ettirilerek iki adet nokta kalıncaya kadar devam ettirilir. Kalan son iki nokta ile de bir adet doğru elde edilir. Sonuçta, oluşabilecek doğruların sayısı;

$(n-1)+(n-2)+..+2+1=\frac{n.(n-1)}{2}$ ile verilir. Buradan da C(n,2) elde edilir. ♦

**Ön teorem :** Kenar sayısı n olan dış bükey poligonun köşegen sayısı $\frac{n.(n-3)}{2}$ ile verilir.

**Kanıt :** Varsalım ki, düzlemde üçü aynı doğrultuda olmayan n adet nokta alalım. Noktalardan geçen doğru sayısının C(n,2) olacağını biliyoruz. Kenar doğruları hariç tutulursa elde edilecek sayı köşegen sayısını verecektir. Yani, n adet kenar sayısı hariç tutulur ise,

$C(n,2) - n = \frac{n.(n-1)}{2} - n = \frac{n.(n-3)}{2}$ elde edilir. Bulunan değer köşegen sayısını ifade eder.♦

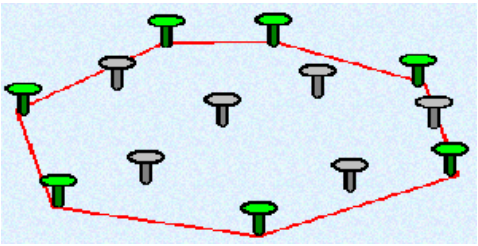
### 1.3. Dış Bükey Kabuk

Düzlemdeki S bölgesi içinde alınacak $p_1$ ve $p_2$ gibi her hangi iki noktayı birleştiren doğru parçası tamamen S bölgesi içinde kalıyorsa S bölgesi dış bükeydir (Bkz. Şekil 1.7.).


Şekil 1.7. Dış bükey, iç bükey bölge

Bütün poligonlar dış bükey değildir. İç bükey olan poligonun köşe noktalarını kullanarak yeni bir dış bükey poligon elde edebiliriz. Elde edilecek dış

bükey poligona dış bükey kabuk denmektedir. Aslında, dış bükey kabuğu düz bir zemine çakılı olan çivilerin etrafını sarmalayan lastik bir ipliğe de benzetebiliriz (Bkz. Şekil 1.8.) (Rasit).

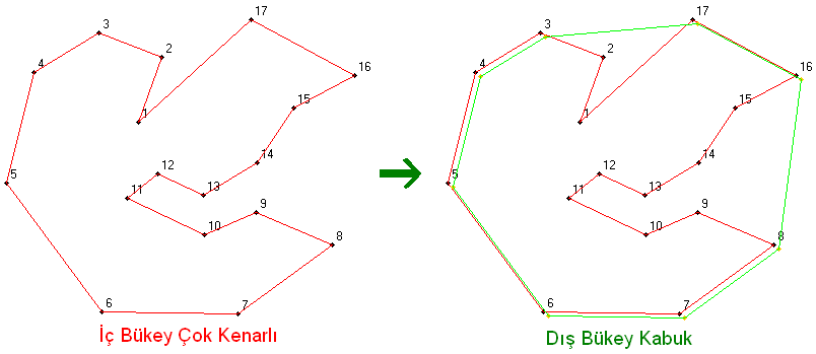


Şekil 1.8. Çivilerin etrafını sarmalayan lastik ip

Düzlemdeki P poligonunu kapsayan dış bükey poligonların en küçüğüne P poligonunun dış bükey kabuğu denir ve CH(P) ile gösterilir.

Başka bir ifade ile, düzlemde sınırlı sayıda noktadan oluşan P poligonunu kapsayan en küçük dış bükey Q poligonu dış bükey kabuktur. Burada en küçük ile kastedilen şey Q'dan farklı ve Q tarafından kapsanan Q` gibi $Q \supset Q` \supset P$ şartını sağlayacak şekilde ikinci bir dış bükey poligon var olmamasıdır (O'Rourke, sf:65).

Tanımdan da anlaşılacağı üzere dış bükey kabuk bir tekdir. Ayrıca, bir dış bükey poligonun dış bükey kabuğu yine kendisidir (Bkz. Şekil 1.9.).




Şekil 1.9. Dış bükey kabuk

Dış bükey kabuk bulma algoritmalarını incelemeye başlamadan önce düzlemdeki sınırlı sayıda noktadan oluşan S poligonunun uç noktalarını inceleyelim. Uç noktalar poligonun en sağ ve en soldaki noktaları ile en üst ve en

alttaki noktalarıdır. Poligonun noktalarının y değeri en küçük olan noktası, en alt noktadır. En küçük y değerli nokta sayısı birden fazla da olabilir. Uç noktaların dış bükey kabukta olacağı açıktır.

Şimdi, dış bükey kabuk bulma algoritmalarını inceleyelim.

### 1.3.1. Dış bükey kabuk bulma algoritmaları

Dış bükey kabuk bulmak amacıyla en çok kullanılan algoritmalar Paket Sarma (Gift Wrap) Algoritması, Graham'ın Tarama (Graham's Scan) Algoritması ve Hızlı Kabuk (Quick Hull) Algoritmalarıdır (Sunday)(Lambert). Çizelge 1.1. de algoritmaların çalıma hızları verilmiştir. n nokta sayısını, h dış bükey kabuk üzerindeki nokta sayısını ifade etmektedir.

Çizelge 1.1. : Dış bükey kabuk bulma algoritmalarının çalışma hızları.

| Algoritma | Çalışma Hızı | En Kötü Durum |
|---|---|---|
| Paket Sarma Algoritması | O(n h) | $O(n^2)$ |
| Graham'ın Tarama Algoritması | O(n.log n) | O(n.log n) |
| Hızlı Kabuk Algoritması | O(n.log n) | $O(n^2)$ |

Önce, dış bükey kabuk bulma algoritmalarından Paket Sarma Algoritmasını inceleyelim.

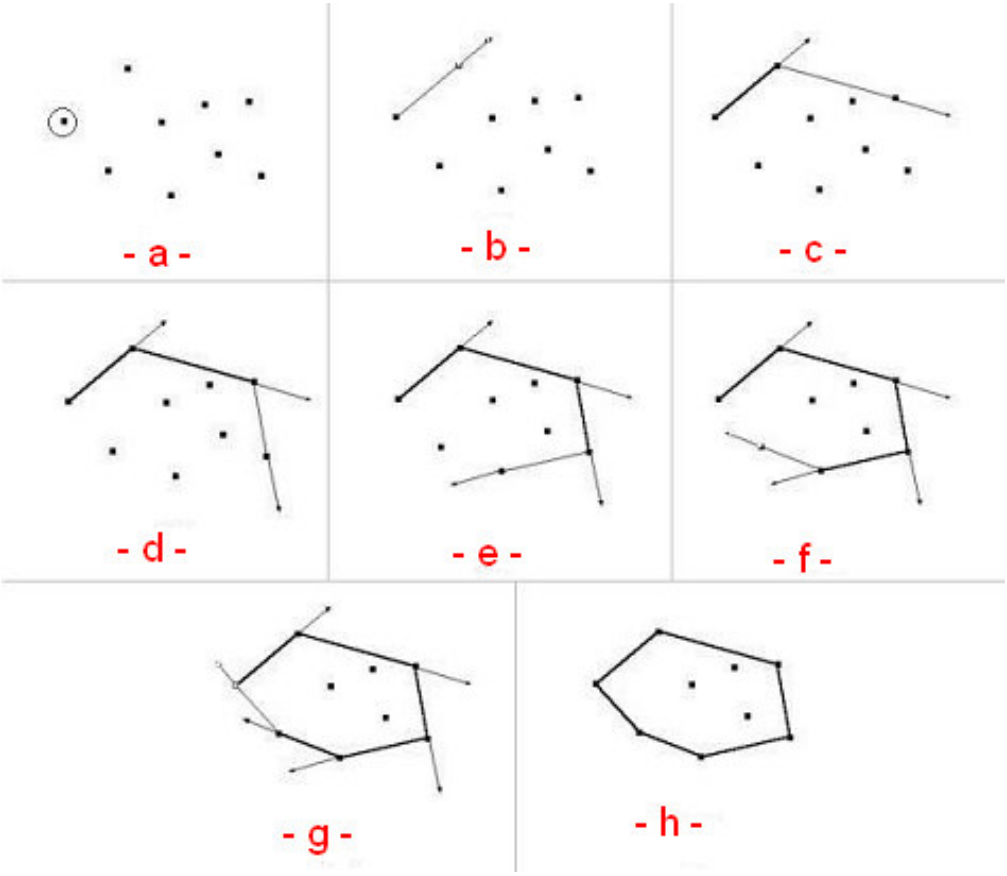
#### 1.3.1.1. Paket Sarma Algoritması

Jarvis March algoritması olarak ta bilinir. Paket Sarma Algoritması bir adet uç nokta bulunduktan sonra en solda ya da en dışta kalan noktayı bulacak şekilde noktaların bulunduğu bölge etrafında dolaşma mantığına dayanmaktadır.

Algoritmanın çalışma mantığını Şekil 1.10 üzerinden ifade edelim. Köşe noktalarının uç noktalarından x-düzlemine göre en küçük x değerli köşe noktası belirlenip dış bükey kabuğun başlangıç noktası olarak kabul edilir (Bkz. Şekil 1.10.a). Sonra, damga sırasına göre poligonun ilk köşesi hedef nokta olarak belirlenir. Hedef nokta ile başlangıç noktası bir doğru ile birleştirilir. Kullanılan noktalar hariç bütün noktaların, oluşturulan doğruya göre konumları incelenir. Eğer

kontrol edilen nokta elde edilen doğrunun sağ tarafında ya da arka tarafında ya da doğru üzerinde ise, damga sırasına göre bir sonraki köşe kontrol edilir. Eğer kontrol edilen nokta doğrunun sol ya da ön tarafında bulunuyor ise, kontrol edilen nokta hedef nokta olarak atanır ve yeni hedef noktaya göre tekrar doğru oluşturulur. Kalan noktalar aynı şekilde kontrol edilir. Bütün noktalar incelendiğinde en son kalan hedef nokta dış bükey kabuğun elemanı olarak listeye eklenir (Bkz. Şekil 1.10.b). Sonra, yeni eklenen nokta başlangıç noktası ve damga sırasına göre kalan noktalardan ilk nokta hedef nokta olarak belirlenir ve süreç bu şekilde devam ettirilir (Bkz. Şekil 1.10.c, d, e, f ve g). Hedef nokta bulunan dış bükey kabuğun ilk köşe noktası olduğunda süreç tamamlanmış olur (Bkz. Şekil 1.10.g ve h) (Smid).

Dış bükey kabuk bulunacak poligona ait köşe noktalarının 1 den n'e kadar damgalanmış olduğunu varsayarsak Paket Sarma Algoritmasının sözde kodları şöyledir:




Şekil 1.10. Paket Sarma Algoritması

**Algoritma**

Paket Sarma Algoritması

Girdi : Poligonun bütün noktalarını içeren noktalar küme listesi

Çıktı : Poligonun dış bükey kabuğunda bulunan noktalarının küme listesi

```
min. x değerli köşeyi bul ve hedef köşe olarak ata
for  w1=1 → n  {
  w1 deki köşe ile hedef köşeyi yer değiştir
  w1 deki köşeyi dış bükey poligona ekle
  w1+1 deki köşeyi yeni hedef köşe olarak ata
  for w2=w1+2 → n  {
       w1 ile hedefteki köşeyi bir doğru ile birleştir
       w2 deki köşenin doğruya göre durumu bul
       if solda ya da  önde ise
            w2 deki köşeyi yeni hedef köşe olarak ata
  }  // Döngü sonu ( w2=w1+2 → n için )
  if hedef köşe  w1 ise işleme son ver  //ilk köşe mi?
} // Döngü sonu ( w1=1 → n  için )
```

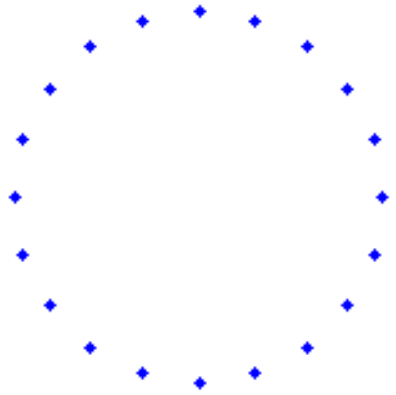
Algoritmada i'inci köşe ile $p_{wi}$ köşesi ifade edilmektedir.

Algoritmanın çalışma hızını bulmak istersek; min. x değerin bulunması için gerekli zaman miktarı O(n) türündendir. İkinci köşenin bulunması için gerekli zaman miktarı da O(n) türündendir ve aynı şekilde üçüncü köşenin bulunması için gerekli zaman miktarı da O(n) türündendir. Süreç bu şekilde devam eder. İlk köşe aranan köşe olduğunda süreç biteceği için dış bükey kabuk köşe sayısını h adet kabul edersek gerekli zaman miktarı

$$O\left(\sum_{k=1}^{h+1} n\right) = O\left(n(h+1)\right) = O(nh)$$

ile verilir.

Bu arada h değerinin 3 ile n arası olduğu unutulmamalıdır. Noktaların konumu nedeni ile en kötü durum için h=n olabilir ki bu durumda algoritma hızı $O(n^2)$ olur. Örneğin, noktalar daire şeklinde sıralanmış ise $O(n^2)$ hızı geçerli olur (Bkz. Şekil 1.11.).



Şekil 1.11. Paket Sarma Algoritması için en kötü durum

Dış bükey kabuk bulma algoritmalarından biri diğeri de Graham'ın Tarama Algoritmasıdır.

#### 1.3.1.2. Graham'ın Tarama Algoritması

Köşe noktalarının y-düzlemine göre en küçük y değerli uç köşe noktası belirlenip bir numaralı damga olarak atanır (Bkz. Şekil 1.12.a). Belirlenen köşe ile diğer köşelere doğrular çizilir ve x-eksenine yaptığı açıları bulunur ve en küçük açılı noktadan büyüğe doğru damga numaraları tekrar düzenlenir (Bkz. Şekil 1.12.b). İlk üç nokta dış bükey kabuk olacak şekilde belirlenir. Sonra, damga sırasına göre sırası ile birer nokta alınır, dış bükey kabuktaki son iki nokta ile aralarındaki açı kontrol edilir. Açı 180° den büyük ise dış bükey kabuktaki son nokta çıkartılarak yeni nokta kabuğa eklenir (Bkz. Şekil 1.12.c ve d). Bu süreç bütün noktalar incelenene kadar devam ettirilir.

Graham'ın Tarama Algoritmasının sözde kodları şöyledir:

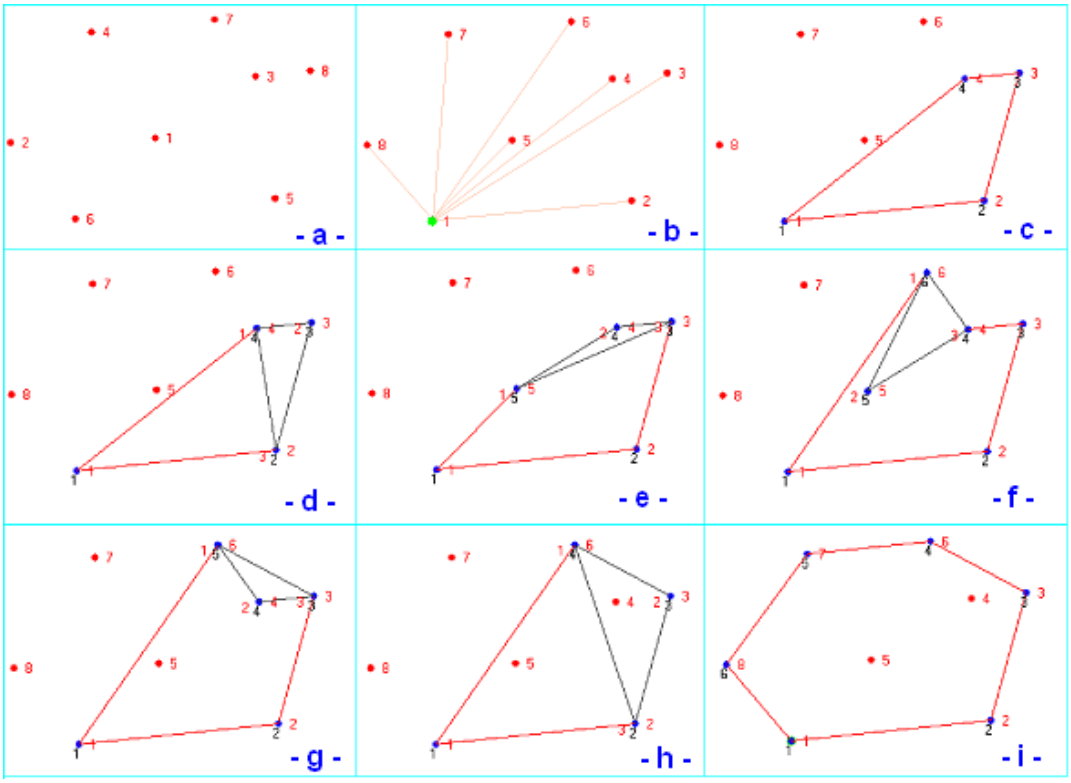
**Algoritma**

Graham'ın Tarama Algoritması

Girdi : Poligonun bütün noktalarını içeren noktalar küme listesi

Çıktı : Poligonun dış bükey kabuğunda bulunan noktalarının küme listesi

```
min. y değerli köşeyi bul ve ilk köşe ile yer değiştir
Köşelerin damgası numaraları açılarına göre sıralanır
İlk üç köşeyi listeye ekle
for  w1=3 → n  {
  Listedeki son 2 köşe ile w1 arasındaki açıyı bul
  if açı 180^0 büyük ise {
       Listeden son köşeyi sil
       w1 deki köşeyi listeye ekle
  }   // if  sonu  (  açı > 180^0  için )
}   // Döngü  sonu  (  w1=3 → n  için)
```




Şekil 1.12. Graham'ın Tarama Algoritması

Algoritmanın çalışma hızını bulmak istersek; min. y değerli köşenin bulunması için gerekli zaman miktarı O(n) türündendir. Ayrıca, köşelerin damga sıra numaranın min. y değerli köşe ile köşelerin açılarının küçükten büyüğe sıralanma hızı O(n.log n) dir. Çünkü, sıralama algoritmalarının en hızlısının hızı O(n.log n) dir. Köşeler sırası ile kullanılacağı için her bir köşe bir kere kullanılır. Dolayısı ile algoritmanın çalışma hızı O(n.log n) türündendir.

#### 1.3.1.3. Hızlı Kabuk Algoritması

İlk önce düzlemdeki sınırlı sayıdaki noktalar kümesinin en küçük x değerli ve en büyük x değerli $P_a$, $P_b$ gibi iki noktası bulunur. Bulunan $P_a$, $P_b$ noktaları dış bükey kabuğun birer elemanı olarak kümeye eklenir. Bulunan $P_a$, $P_b$ doğrularını birleştiren doğrunun sağ ve solunda bulunan noktalardan en uzak olan iki nokta daha bulunur ve kümeye eklenir. Bulunan dört nokta ile saat yönünün tersi yönde hareket ederek her doğrunun sağında kalan en uzak nokta bulunarak dış bükey kümeye eklenerek işleme devam edilir. Dış bölgede nokta kalmayacak şekilde işleme devam edilir ise elde edilen küme ile dış bükey kabuk bulunmuş olur (O'Rourke, sf:71) (Skiena).

Hızlı Kabuk Algoritmasının sözde kodları şöyledir.

**Algoritma**

Hızlı Kabuk Algoritması

Girdi : Poligonun bütün noktalarını içeren noktalar küme listesi ve doğruyu tanımlayan 2 adet nokta

Çıktı : Poligonun dış bükey kabuğunda bulunan noktalarının küme listesi

```
function hizliKabuk(a, b, S) {
  if S boş küme
       return boş
  else if S boş küme değil {
       c = ab ye en uzak nokta
       A kümesi = (a,c) doğrusunun sağında kalan noktalar
```

```
            B kümesi = (c,b) doğrusunun sağında kalan noktalar
            return hizliKabuk(a,c,A) + c + hizliKabuk(c,b,B)
        }  // else if sonu
    }  // fonksiyon sonu
```

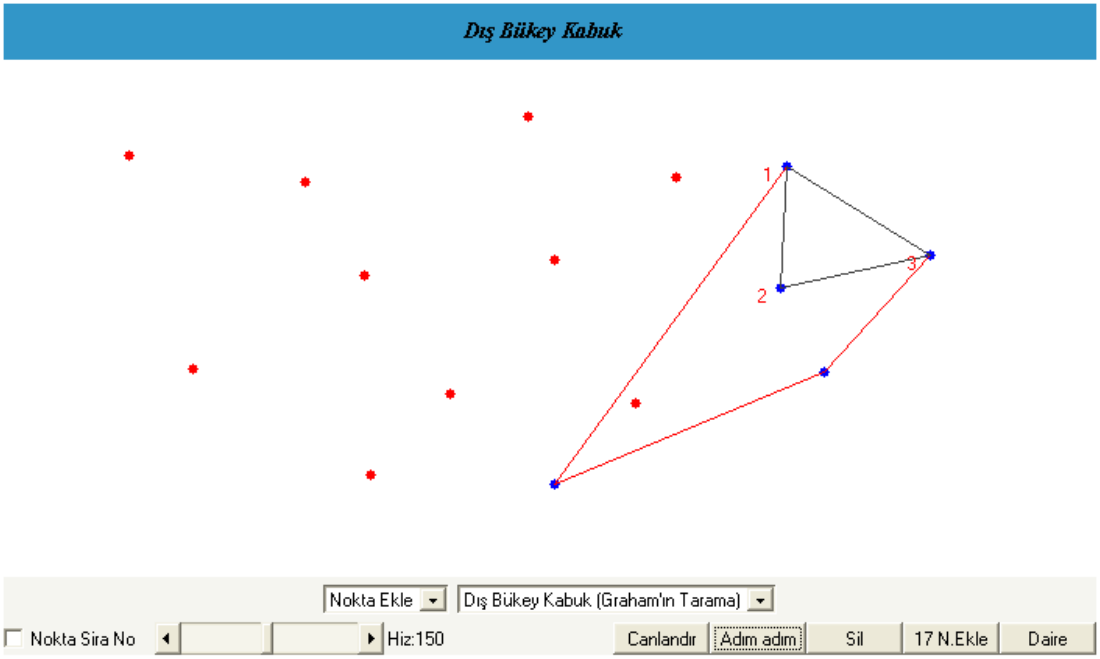
Bu algoritma öz yinelemelidir.

Eleman sayısı n olan S kümesi için algoritmanın çalışma hızını bulmak istersek; Kümenin uç noktaları olan x ve y uç noktalarının bulunması ve S kümesinin $S_1$ ve $S_2$ alt kümelerine ayrılması için gerekli zaman O(n) türündendir. x ve y noktalarını birleştiren doğru parçasına en uzak nokta c olsun. $S_1$ kümesinin eleman sayısı α ve $S_2$ kümesinin eleman sayısı β olsun. Bulunan c noktası $S_1$ ve $S_2$ alt kümelerine dahil olmadığı için $\alpha + \beta <= n - 1$ dir, Bu arada c noktasının bulunması için gereken zaman miktarı O(n) türündendir. Kümenin tamamı için programın çalışma hızını T(n) ile tanımlarsak alt kümeler için hızı T(α ) ve T(β ) olarak tanımlayabiliriz. Bu durumda programın çalışma hızı $T(n) = T(\alpha)+O(n)+T(\beta)$ olur. Program öz yinelemeli çalıştığı için iki alt kümenin çalışması için $T(\alpha)+T(\beta) = O(\log n)$ türündendir. Bu durumda $T(n)=O(n.\log n)$ olur. Yani, algoritmanın çalışma hızı $O(n.\log n)$ dir.

Programın öz yinelemeli çalışmasından dolayı en kötü durumda n adet nokta için n adet yineleme çalışacağından programın çalışma hızı $O(n^2)$ bulunur (Bkz. Şekil 1.11.) (O'Rourke, sf:71). ■

### 1.3.2. Dış bükey kabuk ağ uygulaması

Algoritmaların anlaşılabilir olması için Java applet hazırlanmıştır (Bkz. Şekil 1.13.). Hazırlanan Java uygulamasında girişi yapılan noktaların dış bükey kabuğunun bulunması görselleştirilmiştir. Dış bükey kabuk bulma algoritmalarından Graham'ın Tarama Algoritması ve Paket Sarma Algoritmaları kullanılmıştır. Algoritmaların daha iyi kavranabilmesi amacıyla algoritmaları adım adım çalıştıran "Adım adım" düğmesi Java uygulamasına eklenmiştir. Ayrıca, "Canlandır" düğmesiyle de algoritmanın adım adım çalıştırılması sağlanmış ve her adımın daha iyi gözlemlenebilmesi için algoritmanın çalışması belli bir süre için durdurulmakta, süre bitiminde bir sonraki adıma otomatik geçilmektedir. Dış bükey kabuk bulunması ile canlandırma sona erdirilmektedir.




Şekil 1.13. Dış bükey kabuk bulma algoritmaları Java uygulaması

Poligonları ve dış bükey kabuk bulma algoritmalarını inceledik. Şimdi, poligonların üçgenleştirilmesini inceleyeceğiz.

### 1.4. Üçgenleştirmeye Giriş

Poligonlar bölünerek alt poligonlar elde edilebilir. Elde edilebilecek alt poligonların en küçüğü üçgendir. Başka biri ifade ile, poligon basitleştirmesi işleminde kullanılabilecek en küçük poligon üçgendir. Bu nedenle yapılan basitleştirme işlemine üçgenleştirme de denmektedir. Kare ya da dikdörtgen kullanılabilir, fakat bu nesnelerde köşegenleri yardımı ile iki adet üçgene bölünebilecektir. Daire kullanılmamasının sebebi ise çizimlerde daireler arasında kalacak boşlukların sebep olabileceği görüntüdeki kayıpları neden olarak gösterebiliriz.

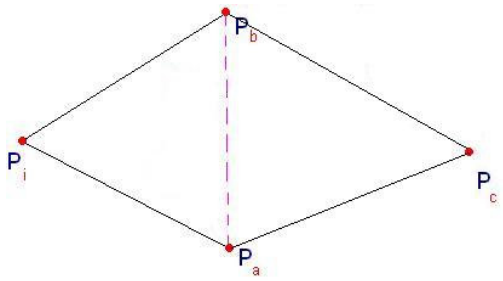
Öncelikler poligonlar ve köşegenleri ile ilgili bazı ön teoremleri inceleyip, üçgenleştirme kavramının nasıl ortaya çıktığını göreceğiz.

**Ön teorem :** Bütün poligonlar en az bir adet dış bükey köşeye sahiptir (O'Rourke, sf:11).

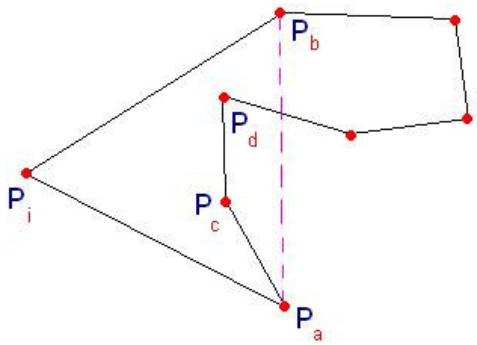
**Kanıt :** Poligona ait noktalar kümesi $P=\{P_0, P_1, .., P_{n-1}\}$ olsun. Kümedeki noktalardan en küçük x değere sahip olan köşelerden her hangi bir köşesi $P_i$ olsun. Seçilen $P_i$ köşesine komşu köşeler $P_a$ ve $P_b$ olsun. Komşu köşelerinin x değerleri $P_i$ nin x değerlerine eşit dahi olsa $[P_aP_i]$ ile $[P_iP_b]$ doğru parçalarının arasındaki açı $180^0$ den küçük ya da eşit olacaktır. Dolayısı ile, $P_i$ köşesi dış bükeydir (Bkz. Şekil 1.14.) ((Bkz. Şekil 1.15.).♦

**Ön teorem :** Köşe sayısı dört ve dörtten fazla olan bütün poligonlar en az bir adet köşegene sahiptir (Meisters Lemma) (O'Rourke, sf:12).

**Kanıt :** Poligona ait noktalar kümesi $\{P_0, P_1, .., P_{n-1}\}$ olsun. Poligonun en küçük x değerine sahip dış bükey köşe $P_i$ olsun. Seçilen $P_i$ köşesine komşu köşeler $P_a$ ve $P_b$ olsun. $[P_aP_b]$ doğru parçası eğer köşegen ise kanıt biter (Bkz. Şekil 1.14.). Eğer, çizilen $[P_aP_b]$ doğrusu köşegen değil ise; $P_aP_iP_b$ üçgensel bölge içinde en az bir $P_c$ gibi bir köşe noktası mevcuttur (Bkz. Şekil 1.15.). Bulunan $P_c$ köşesi ile $P_i$ köşesini birleştiren doğru köşegense kanıt biter. Çizilen $[P_cP_i]$ doğrusu köşegen değilse $P_aP_iP_c$ üçgensel bölgesinde en az bir $P_{c1}$ gibi bir köşe ile $P_bP_iP_c$ üçgensel bölgesinde de aynı şekilde en az bir $P_{c2}$ gibi bir köşe bulunacaktır. Bulunan her bir köşeyle $P_i$ köşesi birleştirilir ve köşegen olup olmadığı kontrol edilir. Süreç bu şekilde devam ettirilirse üçgensel bölgede kalan noktalardan biri mutlaka köşegen olacaktır. ♦




Şekil 1.14. Dış bükey köşe




Şekil 1.15. Dış bükey köşe

Dikkat edilecek olunursa çizilen her bir köşegenle, poligonun iki alt poligona bölündüğü görülecektir. Elde edilen her alt poligon için benzer süreç uygulandığında yeni alt poligonlar elde edilecektir. Poligonların alt poligonları üretildikçe her bölünmede alt poligonların köşe sayısı daha da azalmaktadır. Oluşan alt poligonların köşe sayısı üç ise üçgen elde edilmiş olur ve daha fazla bölünme gerçekleşemez.

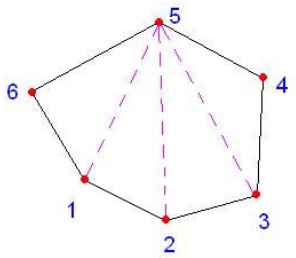
### 1.4.1. Üçgenleştirme tanımı

Düzlemde bütün poligonlar köşegenleri yardımıyla üçgenlere bölünebilir ve bu işleme poligon üçgenleştirmesi denir.

**Ön teorem :** Kenar sayısı n adet olan dış bükey poligonlar her hangi bir köşesinden çizilecek köşegenleri yardımıyla (n – 2) adet üçgene bölünebilir (Vlasic, Sf:4)(Suri).

**Kanıt :** Dış bükey poligonun her hangi bir köşesi seçilir ve seçilen köşeyle kalan $(n - 1)$ köşeye $(n - 1)$ adet doğru çizilir (Bkz. Şekil 1.16.). Doğrulardan iki adedi seçili noktaya komşu olan iki nokta ile çizildiği için kenar doğrusudur ve iki adet kenar doğrusu hariç tutulursa $(n - 3)$ adet köşegen elde edilmiş olur. Köşegen sayısından bir fazla üçgen oluşacağı için $(n-2)$ adet üçgen elde edilir. ♦

Dış bükey poligonun üçgenleştirmesinin çok kolay olduğu açıktır. Dış bükey olmayan poligonların üçgenleştirilmesi içinde benzer kurallar kullanılacak olup, çizilen köşegenlerin kenarları kesmemesi ya da köşegenin poligonun dış bölgesinde kalıp kalmadığına dikkat edilmelidir.




Şekil 1.16. Dış bükey poligonun köşegenleri

### 1.4.2. Üçgenleştirme teoremi

Bütün poligonlar köşegenleri sayesinde üçgenlerine ayrılabilirdir (O'Rourke, sf:12).

**Kanıt:** Poligona ait noktalar kümesi $\{P_0, P_1, .., P_{n-1}\}$ olsun. Köşe sayısı üç ise kanıt aşikardır. Köşe sayısı dört ise karşılıklı iki köşeden çizilebilecek iki doğrudan en az bir tanesi köşegendir ve dörtgeni iki üçgene böler. Köşe sayısı dörtten fazla ise; en az bir köşegene sahip olduğunu kanıtlanmıştı. Çizilen köşegen ile poligon iki alt poligona bölünür. Oluşan her alt poligonun da en az bir köşegene sahip olduğu dikkate alınırsa alt poligonlara ait yeni köşegenler ve yeni alt poligonlar elde edileceği görülecektir. Oluşan her bir alt poligonun köşe sayısı üç adet oluncaya dek bölünme işlemine devam edilirse köşe sayısı üçten büyük olan poligon kalmayacaktır. Sonuç olarak, poligon üçgenlerine bölünmüş olacaktır.♦

**Ön Teorem :** Köşe sayısı $n \geq 4$ olan bir poligon $(n - 3)$ köşegen ile $(n - 2)$ üçgene bölünür (O'Rourke, sf:12).

**Kanıt :** Poligona ait noktalar kümesi $\{P_0, P_1, .., P_{k-1}\}$ olsun. Bütün poligonların en az bir adet köşegene sahip olduğunu biliyoruz. Çizilen köşegen ile poligon, köşe sayıları $k_1 \geq 3$ ve $k_2 \geq 3$ olmak üzere iki alt poligona bölünmüş olur. Çizilen köşegenin iki köşe noktasının her iki poligonda da olacağı dikkate alınmalıdır. Bu durumda $k_1 + k_2 = k + 2$ olacaktır.

Kanıtı poligonun köşe sayısı üzerinden tümevarım yöntemiyle yapalım.

Eğer, $n=4$ için köşegen sayısı $n-3=1$ ve üçgen sayısı $n-2=2$ olacağından önerme doğrudur. Varsayalım ki, eğer $n=k_1$ ve $n=k_2$ için Ön teorem doğru olsun. Bu durumda $n=k$ için doğru olup olmadığını kontrol etmemiz yeterli olacaktır.

Varsayımımıza göre $n=k_1$ için alt poligon $(k_1 - 3)$ köşegen ile $(k_1 - 2)$ üçgene bölünür ve $n=k_2$ için alt poligon $(k_2 - 3)$ köşegen ile $(k_2 - 2)$ üçgene bölünür.

İşlem n=k için doğru olup olmadığını kontrol edelim. Poligonun çizilen bir köşegen yardımıyla iki alt poligonun bölündüğünü ve iki köşenin her iki alt poligonda da ortak köşeler olduğunu ve dolayısıyla iki köşeyi birleştiren doğru parçası da ortak kenar olduğunu belirtmiştik. Şimdi, iki alt poligonun birleşiminden oluşacak poligonun köşe sayılarını inceleyelim. Ortak kenarın yeni poligonda köşegen olacağını da göz önüne alırsak elde edilecek köşegen sayısı şu şekilde bulunur:

$$(k_1 - 3) + (k_2 - 3) + 1 = (k_1 + k_2) - 5$$

$$= ( k + 2 ) - 5$$

$$= k - 3$$

$= n - 3$ adet köşegen bulunur.

Yine aynı şekilde, iki poligonun birleşiminden meydana gelen yeni poligonun oluşan üçgen sayısı ise;

$$(k_1 - 2) + (k_2 - 2) = (k_1 + k_2) - 4$$

$$= ( k + 2 ) - 4$$

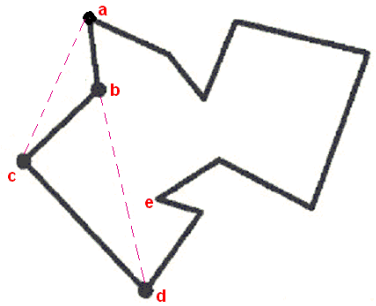
$$= k - 2$$

$= n - 2$ adet üçgendir (Kreveld, sf:47). ♦

**Sonuç :** Nokta sayısı n olan poligonun iç açılarının toplamı $(n-2)\pi$ dir.

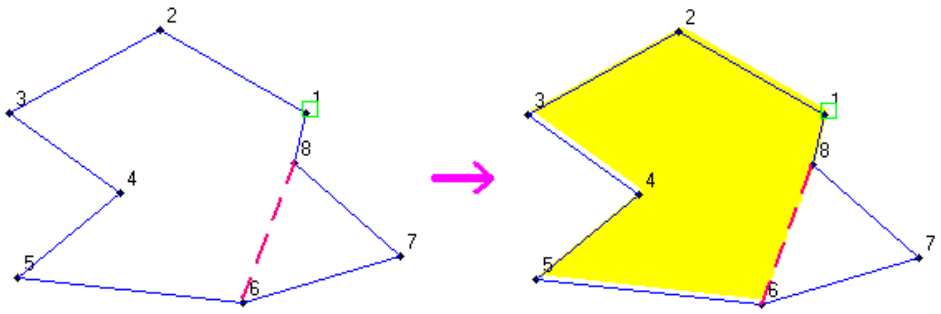
**Kanıt :** Son ifade edilen Ön Teorem'e göre n adet köşeden oluşan poligon $(n-2)$ adet üçgene bölünebilir. Her üçgenin iç açısı $\pi$ olduğuna göre $(n-2)\pi$ sonucu elde edilir (O'Rourke, sf:14). ♦

Kullanılacak köşegen hem poligona ait kenar doğrularını kesmemeli hem de tamamen poligonun iç bölgesinde kalmalıdır. Şekil 1.17'de görüleceği üzere çizilen [ac] köşegeni ile $T_{abc}$ üçgeni oluşturulmakta, fakat $T_{abc}$ üçgeni poligonun iç bölgesinde kalmamaktadır. Dolayısı ile [ac] köşegeni geçersiz köşegendir.

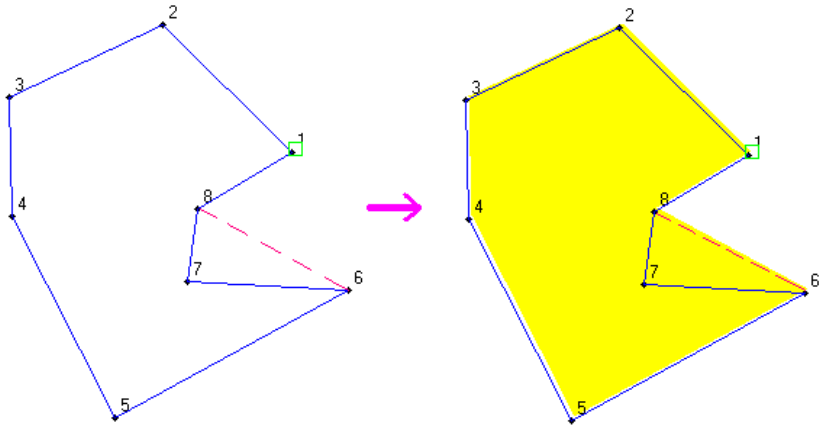



Şekil 1.17.Geçerli ve geçersiz köşegenler

Benzer şekilde, Şekil 1.18.'de görülebileceği gibi altıncı ve sekizinci köşeleri birleştiren köşegen yedinci köşe hariç tutularak oluşturulan poligonun içinde kalırken Şekil 1.19.'de altıncı ve sekizinci köşeleri birleştiren köşegen yedinci köşe hariç tutularak oluşturulan poligonun dışında kalmaktadır. Uygulamaları kodlarken önce köşegenin poligon içinde kalıp kalmadığı kontrol edilmeli ve sonra kenar doğrularını kesip kesmediği kontrol edilmelidir. Köşegenin poligona göre durumunu analiz etmek için şu kontrol yapılabilir: köşegen çizildiğinde oluşacak üçgenin üçüncü köşe noktası poligondan hariç tutularak oluşturulacak yeni poligonun üçüncü köşeyi kapsayıp kapsamadığı kontrol edilebilir. Yeni poligon üçüncü noktayı kapsamıyorsa üçüncü köşe dış bükey köşedir ve köşegen poligonun iç bölgesinde kalıyordur (Bkz. Şekil 1.18.). Eğer, yeni poligon üçüncü noktayı kapsıyorsa üçüncü köşe iç bükey köşedir ve köşegen poligonun dış bölgesinde kalıyordur (Bkz. Şekil 1.19.). Yapılacak bu kontroller, köşegenin diğer kenar çizgilerinin kesip kesmediğinin kontrolü ile tamamlanır.




Şekil 1.18. Geçerli köşegen




Şekil 1.19. Geçersiz köşegen

Ayrıca, Şekil 1.18.'de görüleceği üzere, bir birine komşu üç köşeden ortadaki köşe dış bükey ise köşegen çizilerek poligon iki alt poligona bölünür. Köşe sayısı üç olan alt poligonun üçgen olacağına dikkat edilecek olunursa, poligonda her üç sıralı köşe için orta köşe dış bükey olmak ve iki köşeyi birleştiren köşegen kenar doğrularını kesmemek şartı ile biri üçgen olmak üzere poligon iki alt poligona bölünür. Süreç en son kalan alt poligonun köşe sayısı üç kalıncaya kadar devam ettirilirse poligon üçgenleştirilmiş olur.

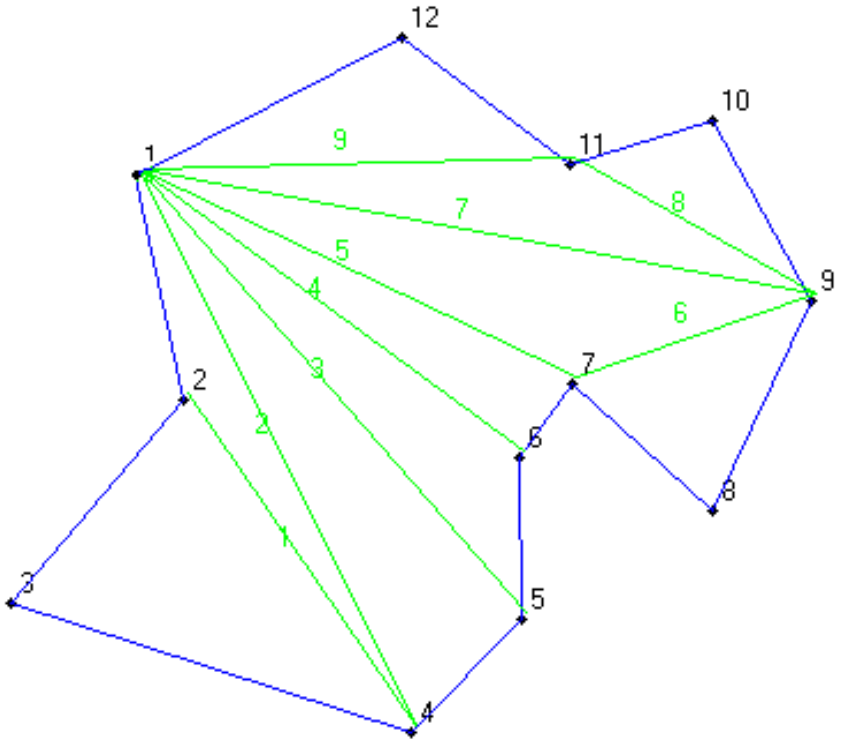
Köşegenler atılarak yapılacak üçgenleştirme yönteminin inceleyelim.

## 1.5. Üçgenleştirme Algoritmaları

### 1.5.1. Sınırları Bölme Algoritması

Bütün poligonlar köşegenleri aracılığı ile üçgenler kopartılarak üçgenleştirilebilir (Mukherjee, sf:12).

Köşe sayısı N olan bir poligonun damga sırası ile komşu üç nokta orta nokta dış bükey olmak şartı ile köşegen kopartılabiliniyorsa poligondan üç noktanın oluşturduğu üçgen kopartılır. Şekil 1.20'de görüleceği üzere $[p_1p_3]$ köşegeni ve $T_{123}$ üçgeni poligon dışında kaldığı için $p_2$, $p_3$ ve $p_4$ köşeleri incelenir. $[p_2p_4]$ köşegeni geçerli olduğu için $T_{234}$ üçgeni atılabilir. $T_{234}$ üçgeni atılınca $p_1$, $p_2$ ve $p_4$ köşelerine ait $[p_1p_4]$ köşegeni geçerli hale gelir ve $[p_1p_4]$ köşegeni ikinci köşegen olarak atılabilir. Yani, $T_{124}$ üçgeni atılmış olur. Şekil 1.20'de köşegenlere verilen damga numarası köşegenlerin poligondan atılış sırasını ifade etmektedir (Stewart).




Şekil 1.20. Sınırları Bölme Algoritması

Aslında, dikkat edilecek olunur ise, $T_{234}$ üçgeni poligondan atılırken poligondan sadece $p_3$ köşesi atılmış gibi olmaktadır. Aynı şekilde $T_{124}$ üçgeni atılırken poligondan sadece $p_2$ köşesi atılmış olmaktadır.

Köşegenleri atma yöntemi ile yapılan üçgenleştirmenin sözde kodlaması şöyledir.

**Algoritma**

Sınırları Bölme Algoritması

Girdi : Damga sıralı poligon listesi

Çıktı : Üç noktalı üçgenler listesi

```
if ( n > 3 ) {
  p1, p2, kuyruğa ekle
  while |P|≥3 do {
     pj kuyruğa ekle  // sıradaki noktayı ekle (j≥3)
     while kosegenmi (P, pi-2 pi ) ise {
          (pi-2 , pi-1 , pi ) üçgeni oluştur
          pi-1 köşesini kuyruktan çıkart
          pi-1 köşesini poligondan çıkart
     } // while döngü sonu ( kosegenMi (P, pi-2 pi ) )
  } // while döngü sonu ( |P|≥3 )
  (pi, p2, p3) üçgeni oluştur // kuyrukta kalan son 3 köşe
} // if sonu
```

Sınırları Bölme Algoritması için bir uygulama örneği Şekil 1.20'de görülmektedir.

Köşe sayısı n olan bir poligonda noktaların kontrol edilmesi için gerekli zaman miktarı O(n) türündendir. Ayrıca, seçilen köşegenin poligonun kenarlarını kesip kesmediğinin kontrol edilmesi için gerekli zaman miktarı da O(n) türündendir. Noktaların kuyruğa atılması, kuyruktan alınması için gerekli zaman miktarı da O(n) türündendir. Sonuç olarak, programın çalışması için gerekli zaman miktarı $O(n^3)$ türündendir.

Köşegenlerin poligonun kenar çizgilerini kesip kesmediğinin kontrol edilmesi gerektiği belirtilmiştir. Bu kontrolün yapılması için gerekli algoritma şu şekilde kodlanır.

**Algoritma**

Girdi: P poligon listesi ile iki adet poligon noktası

Çıktı: yanlış/doğru

```
kosegenmi (P, p_{i-2} , p_i ) {
    for j = 1 → N {
        if ( p_j ≠ p_i ve p_j ≠ p_{i-2} )
           if kesiyor((p_j, p_{j-1} ), (p_{i-2} , p_i ))
             köşegen değil
    } // for döngü sonu
}
```

Basit poligon için üçgenleştirme, poligonu maksimum sayıda birbirini kesişmeyen köşegenler ile bölünmesidir. Genel olarak üçgenleştirme bir tek değildir.

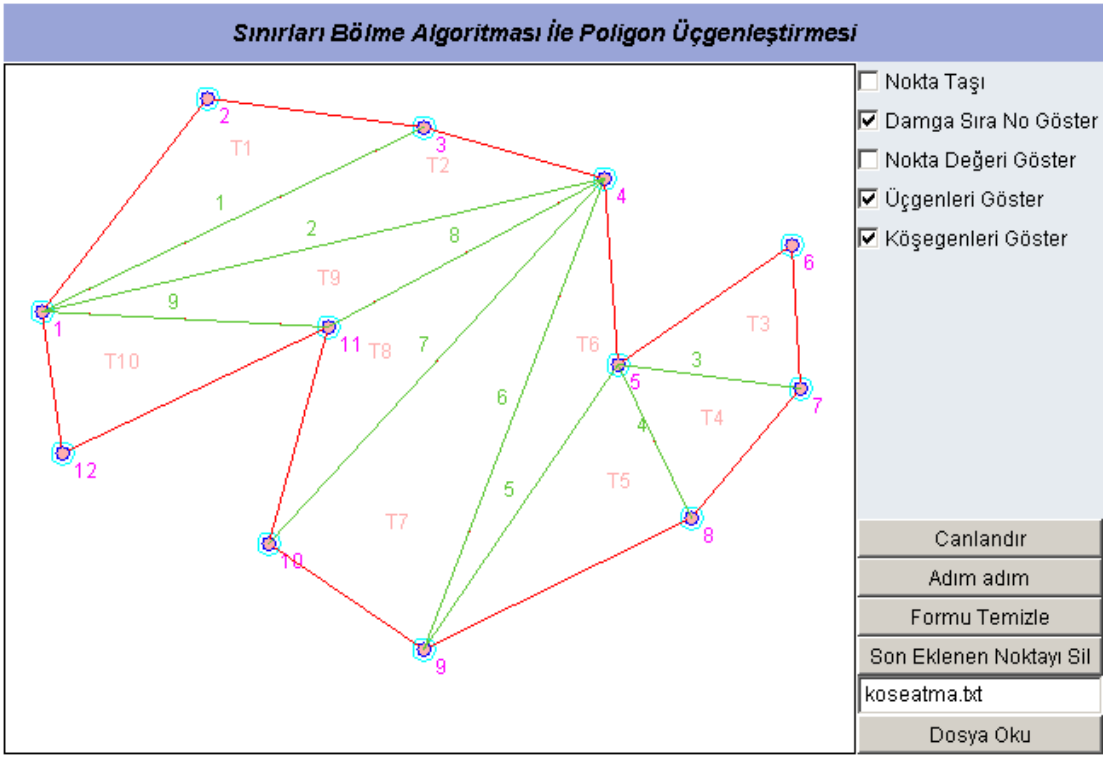
### 1.5.2. Sınırları Bölme Algoritması ağ uygulaması

Hazırlanmış olan ağ uygulamasında nokta giriş işlemi ilk girilen noktanın farenin imleci vasıtasıyla tekrar tıklanması ile sona erer. "Üçgenleri Göster" seçeneği ile üçgenleştirme sonucu görselleştirilir (Bkz. Şekil 1.21.). "Köşegen Göster" seçeneğiyle ise köşegenlerin atılış sırasını gösterir. Ayrıca, uygulamaya eklenen "Adım adım" ve "Canlandır" düğmeleri ile algoritmanın çalışma aşamaları adım adım görselleştirilmiş, bu sayede algoritmanın çalışma mantığının daha iyi kavranması hedeflenmiştir.

**Teorem :** Tüm basit poligonlar üçgenleştirilebilir ve her hangi bir n adet köşeden oluşan basit poligon üçgenleştirilmesinden $(n - 2)$ adet üçgen elde edilir.

**Kanıt :** Bir üçgen üç adet köşeden oluştuğu için $n>3$ olduğunu kabul edebiliriz. Varsayalım ki P poligonu n adet köşe noktasından meydana gelsin. v köşesi en küçük x değerli nokta olsun. w ve u köşeleri de v köşesine komşu köşeler olsun. uw doğru parçası poligon içinde kalıyorsa uw köşegeni poligonun

bir köşegenidir (Bkz. Şekil 1.22.). uw doğru parçası poligon içinde kalmıyorsa v, w, u üçgen bölgesi içerisinde en az bir köşe vardır. v' köşesi v, w, u üçgen bölgesinde kalan ve uw köşegenine en uzak köşe nokta olsun. vv' doğru parçası poligonu kesmez. Sonuç olarak vv' doğru parçası köşegendir (Bkz. Şekil 1.23.).




Şekil 1.21. Sınırları Bölme Algoritması ağ uygulaması

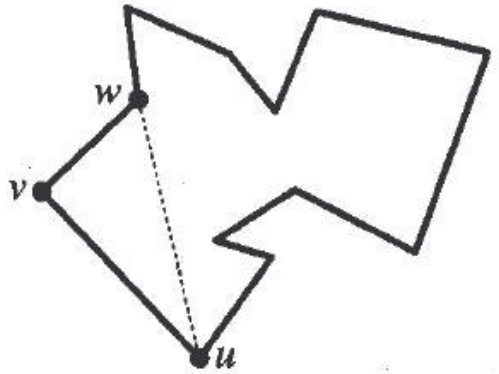
Her poligon en az bir adet köşegene sahiptir ve her bir köşegen poligonu iki alt poligona böler. Yeni oluşan alt poligonlara $P_1$ ve $P_2$ ve köşe sayılarına $m_1$ ve $m_2$ dersek, $m_1<n$ ve $m_2<n$ olur ve köşegene ait köşeler her iki alt poligonda da var olduğu için (iki köşe tekrarlandığı için ) $m_1 + m_2 = n + 2$ elde edilir.

Ayrıca, varsayıma göre $P_1$ poligonu $(m_1 - 2)$ ve $P_2$ poligonu $(m_2 - 2)$ üçgenden meydan gelir.

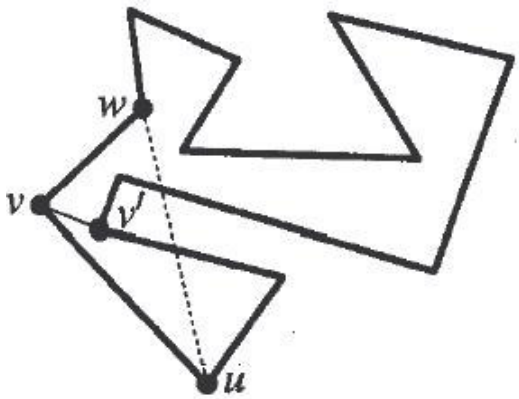
Alt poligonları meydana getiren üçgenlerin sayısı

$(m_1 - 2) + (m_2 - 2) = (m_1+m_2 - 4) = (n+2 - 4) = (n - 2)$ olarak bulunur. ♦ (Ottmann, Sf:9,10,11)

Poligondan atılacak köşenin dış bükey olması gerekmektedir. Şekil 1.23'te görüleceği üzere damga numarası $p_2$ olan köşe iç bükeydir. Bir sonraki köşe olan $p_3$ damga numaralı köşe ise dış bükeydir. $[p_2p_4]$ köşegeni geçerli köşegen olup $T_{234}$ üçgeni atılabilir. $T_{234}$ üçgenini atmak için $p_3$ köşesinin atılmasının yeterli olduğunu belirtmiştik. Atılan $p_3$ köşesi ile $p_2$ köşesinin dış bükey olacağı görülecektir. Yani, dış bükey köşeler atıldıkça iç bükey köşeler de dış bükey köşeye dönüşebilmektedir. Özetle, poligonun damga sırasına göre dış bükey olan köşeleri atılarak üçgenleştirilebilir İç bükey olduğu için atılamayan köşeler, atılan köşeler ile dış bükey olup olmadıkları kontrol edilip, dış bükeye dönüşen köşeler de atılır. Bu yöntemle bütün köşeler atılabilir (Jia).



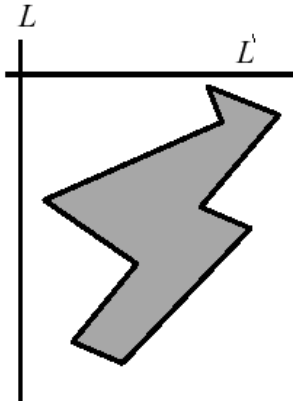

Şekil 1.22. Uç noktaya göre geçerli köşegen




Şekil 1.23. Uç noktaya göre geçersiz köşegen

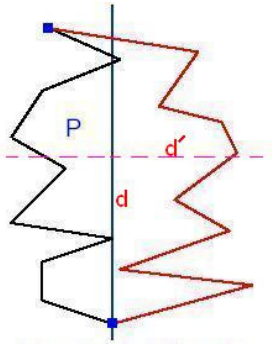
Aynı şekilde, poligon dış bükeyse üçgenleştirme daha kolay ve hızlı olacaktır. Çünkü, dış bükey poligonda her hangi bir köşeden komşu köşelere çizilecek köşegenlerle kolayca üçgenleştirme elde edileceğinden dış bükey poligonlar için üçgenleştirme daha hızlıdır. Aynı şekilde, monoton poligonların da üçgenleştirilmesi monoton olmayan poligonlara göre daha kolay ve hızlı olacaktır.

Bir L doğrusu poligonu iki alt poligona bölsün. L doğrusuna dik olacak şekilde çizilecek L` gibi doğrular alt poligonları sadece tek bir noktada kesiyorsa P poligonu L çizgisine göre monotondur ya da L-monotondur denir (Bkz. Şekil 1.24.).



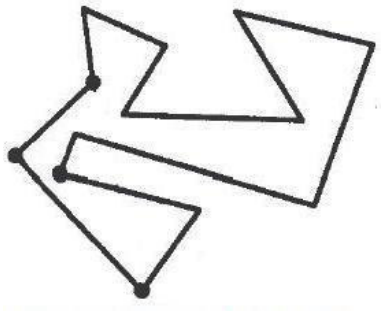

Şekil 1.24. l-Monoton

Aynı şekilde, y eksenine paralel olacak şekilde bir d doğrusuyla poligon iki zincire parçalansın. Her bir zincir d doğrusuna dik çizilecek bir başka doğruyla tek bir noktada kesişiyorsa poligon d doğrusuna göre y-monotondur ya da monotondur denir (Bkz. Şekil 1.25.) (Shewchuk, sf:9)(D'Hondt).




Şekil 1.25. y-Monoton

Ayrıca, poligonu hiçbir doğruya göre monoton yapılamıyorsa poligon monoton değildir (Bkz. Şekil 1.26.).



Şekil 1.26. Hiçbir doğruya göre monoton olmayan poligon

### 1.5.3. Süpürme yöntemi ile monoton poligon üçgenleştirmesi

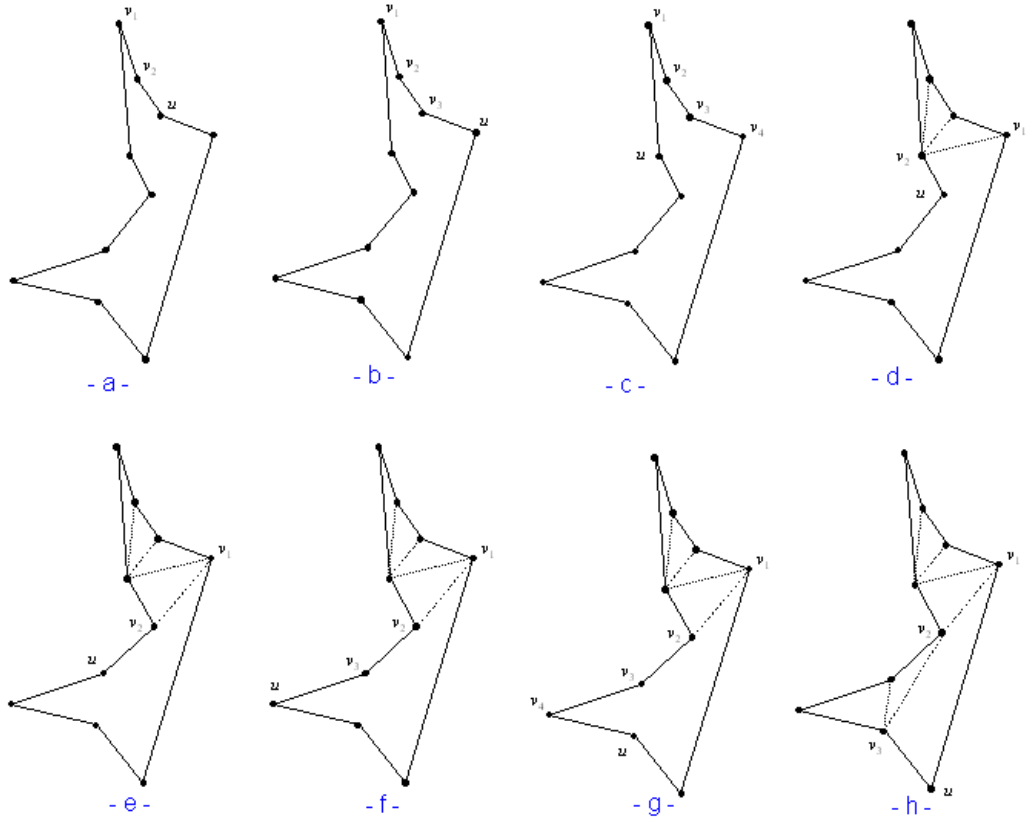
Monoton poligonları dış bükey poligonlar kadar kolay olmasa da şekil itibari ile düzgün ya da sade oldukları için üçgenleştirilmeleri karmaşık poligonlara göre daha kolaydır. Burada uygulanan üçgenleştirme mantığı şöyledir: Poligon y-monotonsa y değeri büyük olan iki köşe kuyruğa atılır. Sonra, y değeri büyük olandan y değeri küçük olana doğru poligonun iki zinciri üzerinde ilerlenir. Her bir nokta için kuyruğa son konmuş olan nokta ile üzerinde bulunan noktanın aynı zincir üzerinde olup olmadığı kontrol edilir (Bkz. Şekil 1.27.a).(Quadros)

i. Noktalar aynı zincir üzerindeyseler kuyruktaki sondan ikinci nokta ile üzerinde bulunulan noktanın birleştirilmesiyle çizilecek köşegenin geçerli olup olmadığı kontrol edilir. Köşegen poligon dışında kalıyorsa geçersizdir. Köşegen geçersizse üzerinde bulunulan nokta kuyruğa son nokta olarak atılır ve bir sonraki noktaya geçilir. Köşegen geçerliyse kuyruktaki son nokta kopartılarak ve üzerinde bulunulan nokta, kopartılan nokta ve kuyruktaki son nokta ile üçgenleştirilir. Kuyruktaki nokta sayısı birden fazla ve üzerinde bulunulan nokta ile kuyruktaki son noktayı birleştiren köşegen geçerli olduğu sürece süreç tekrarlanır (Bkz Şekil 1.27.h). Süreç bittiğinde üzerinde bulunulan nokta kuyruğa son nokta olarak atılır ve bir sonraki noktaya geçilir.

ii. Noktalar farklı zincir üzerindeyseler üzerinde bulunulan nokta ile kuyruktaki son noktadan bir önceki noktayı birleştiren köşegen geçerli ise üzerinde

bulunulan nokta, kuyruktaki son nokta ve kuyruktaki son noktadan bir önceki nokta birleştirilerek üçgenleştirilir, kuyruktaki son noktadan bir önceki nokta kuyruktan atılır. Kuyrukta bir nokta kalıncaya kadar süreç bu şekilde devam ettirilir (Bkz. Şekil 1.27.c ve d). Süreç sonunda üzerinde bulunulan nokta kuyruğa son nokta olacak şekilde atılır.

Son aşamada kuyrukta kalacak iki nokta ile en küçük y değerli son nokta birleştirilerek üçgenleştirilir.




Şekil 1.27. Süpürme yöntemi ile üçgenleştirme

Benzer şekilde, poligon x-monotonsa x değeri küçük olan köşeden başlanarak x değeri büyük olan köşeye doğru ilerlenir. Noktaların ilk ikisi kuyruğa atılır. Sonra, gelecek noktaların aynı zincir üzerinde olup olmadığı kontrol edilir. Aynı zincir üzerinde ise köşegenin geçerli olup olmadığı kontrol edilir. Her işlemde kullanılmayan noktalar kuyruğa atılıp, üçgenleştirilirken alınarak süreç devam ettirilir. Poligon boyunca bir kere gidilip, kalan poligon üzerinden işlem yapıldığı için algoritmanın çalışması hızlıdır (Shewchuk) (Mukherjee) (Ottmann)(Garey).

**Algoritma**

Süpürme Yöntemi ile Üçgenleştirme Algoritması

Girdi : Damga sıralı P poligon listesi

Çıktı : Üç noktalı üçgenler listesi

```
Liste y değerine göre azalan sıra sıralanır.
En büyük y değerli köşe kuyruğa at//S.push(u1);S.push(u2);
for j = 3 → n-1
  uj köşesini işleme koy
  if (uj S kuyruğundaki 2 köşe ile farklı zincirde ise){
    S teki köşeleri çıkart
    Çıkan 2 köşe ile uj köşesi birleştirilir
    //{ Yani, v=S.pop(); diagonal(v,uj); }
    uj-1 ve uj köşelerini S kuyruğuna koy
    //{ Yani, S.push(uj-1 ); ve S.push(uj ); }
  } else { // aynı zincirde ise
     while (S'ten üsten koparın ile uj köşegeni
            P de kalıyor ise ) {
            //{Yani, while kosegenmi (P, S.top, uj) in P}
            kosegenmi (P, S.top, uj);
            S.pop();  // en üst köşe kopartılır
     }  // while döngü sonu
  } // else if sonu
  S.push(sonraki köşe); // S kuyruğuna eklenir
  S.push(uj ); // S kuyruğuna eklenir
  Son kalan üç noktayı al
} // for döngü sonu
```

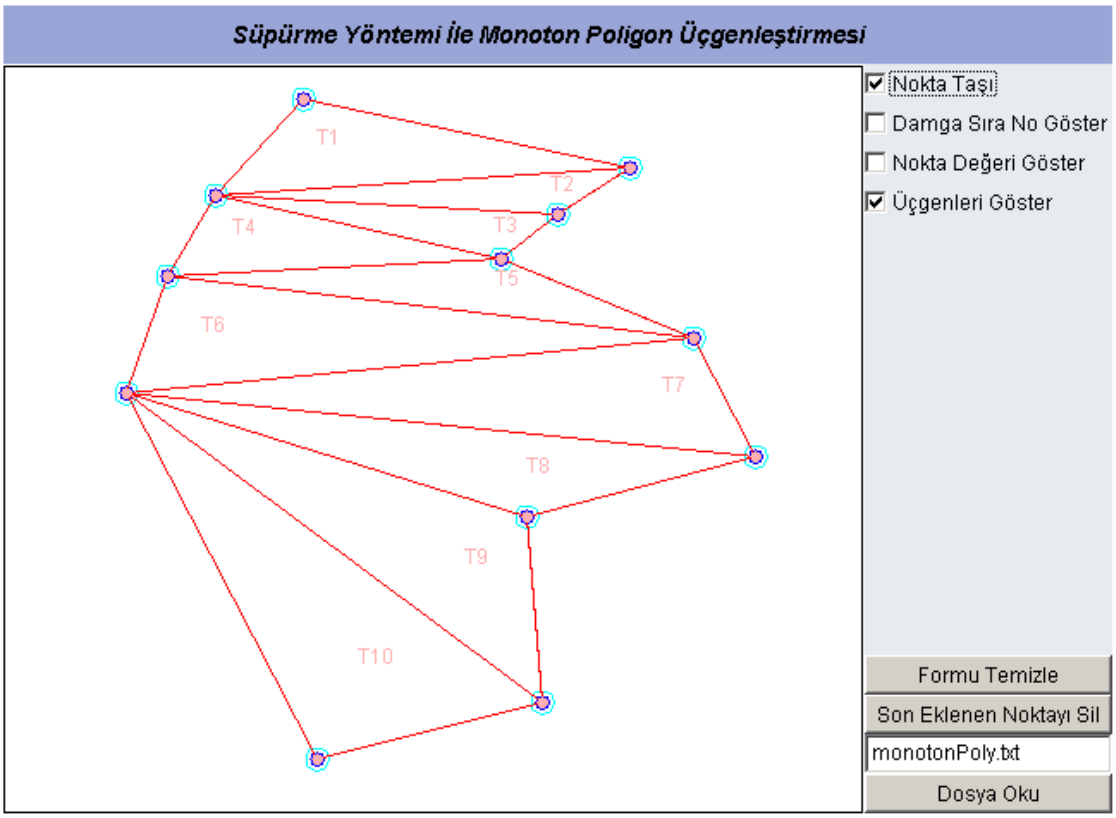
Algoritmanın çalışma hızını inceleyelim.

Köşelerin sıralanması O(n) zamanda yapılır. Sıralanmış liste üzerinde hareket ederken her bir köşe kuyruğa bir kere atılmakta ve kuyruktaki noktalara kuyruktan bir kere koparılmakta ve bir daha kuyruğa geri konmamaktadır. Dolayısıyla algoritmanın çalışma hızı O(n) türündendir (Jia).

Ayrıca, en büyük y değerli köşe bulunurken en büyük y değerine sahip köşe sayısı bir den fazla olabilir. Bu durumda y değeri en büyük olan köşeler arasında x değeri en büyük olan seçilir.

### 1.5.4. Süpürme yöntemi ile üçgenleştirme ağ uygulaması

Hazırlanmış olan ağ uygulamasında nokta giriş işlemi ilk girilen noktanın farenin imleci vasıtasıyla tekrar tıklanması ile sona erer. “Üçgen” seçeneği ile süpürme yöntemi ile monoton üçgenleştirme algoritması çalıştırılır ve üçgenleştirme sonucu oluşan üçgenler görüntülenir (Bkz. Şekil 1.28.). Bu Java applet’i x-monoton ve y-monoton poligonların üçgenleştirilmesi amacıyla hazırlanmıştır.

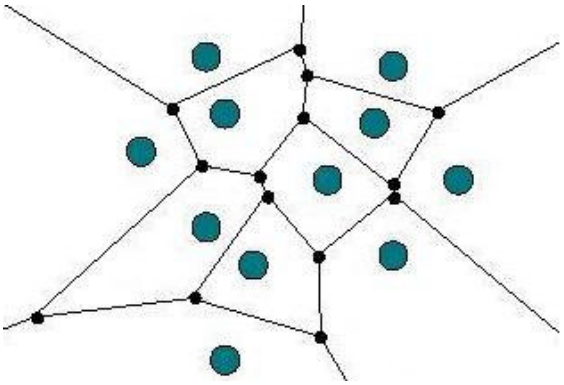



Şekil 1.28. Süpürme yöntemi ile üçgenleştirme ağ uygulaması

## 2. VORONOI ÇİZGELERİ VE DELAUNAY ÜÇGENLEŞTİRMESİ

Bir şehirde, bir bölgede hizmet verecek personel için görev dağılımı yapıldığını kabul edelim. Bir örnek ile ifade etmek istersek, Ankara'da su sayaçlarını okuyan şirket, okuma memurlarının hangi adreslere gideceğinin planını yapıp personele bildirir. Su sayaçları okunacağı zaman her bir personel sadece kendisine ait bölge içerisindeki adreslere, binalara uğrar ve okuma işini yapar. Hiçbir personel kendi bölgesi dışında bir bölgeye geçmez. Bölgenin belirlenmesinde iş performansın en iyi olması hedeflenir. Ayrıca, görev karmaşası olmaması için her bir alt bölgede ya da hücrede sadece ve sadece bir personel istihdam edilmektedir.

Şehirlerde elektrik dağıtımını yapacak trafoların şehir bölgesinde yerleştirilmesi ve her bir trafonun içinde bulunduğu hücre ya da alt bölgede hizmet vermesi, su pompalarının şehir üzerinde dağılımları, itfaiye hizmetlerinin dağılımları, .. gibi. Bu türden örnekleri geliştirebiliriz ( D'Hondt2). Dikkat edilecek olunursa, bölge hücrelere bölünmekte ve her bir hücreye sadece bir eleman yerleştirilmektedir.

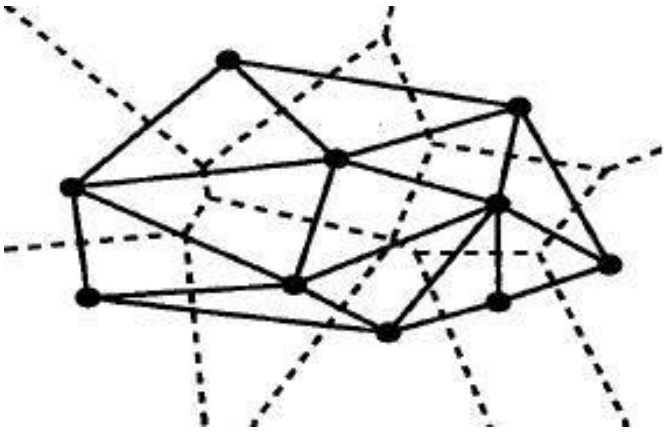


Şekil 2.1. Voronoi çizgeleri

Düzlemde yer alan sonlu nokta kümesine ait herhangi bir noktaya, kümedeki diğer noktalardan daha yakın konumda bulunan düzlem noktalarının geometrik yerine, o noktanın Voronoi Poligonu denilmektedir. Kümedeki tüm noktaların Voronoi Poligonların birleşimi, o kümenin Voronoi çizgesini oluşturur. (Yanalak,1997).

Ayrıca, üçgenleştirmeden elde edilen üçgenlerin üç köşesinden geçecek şekilde çizilecek çevrimsel çemberlerin içerisinde hiçbir köşe kalmayacak şekilde

tanımlanmış ise bu üçgenleştirmeye Delaunay üçgenleştirmesi denir (Bkz. Şekil 2.2.).



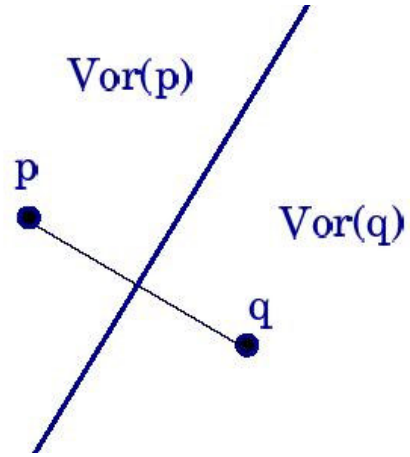
Şekil 2.2. Voronoi çizgeleri ve Delaunay üçgenleştirmesi

Matematiksel olarak Voronoi çizgeleri ve Delaunay üçgenleştirmesi birbirlerini tamamlar. Yani, Voronoi çizgelerinden Delaunay üçgenleştirmesi elde edilebileceği gibi Delaunay üçgenleştirmesinde de Voronoi çizgeleri elde edilebilir. Herhangi bir T Delaunay üçgeninin üç köşesi üç farklı Voronoi bölgesi içinde kalmaktadır. Aynı zamanda, T Delaunay üçgeninin çevrimsel çemberinin merkezi Voronoi çizgelerinin köşe noktalarını tanımlar (Bildirici).

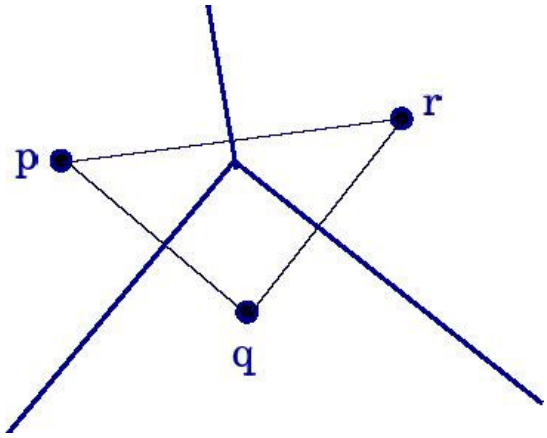
Voronoi düzleminde $p_i$ noktasını kapsayan hücreyi ifade etmek için Vor($p_i$) gösterimi kullanılacaktır. Düzlemi kaplayan Vor($p_i$)'lerin birleşimi ise Vor(P) ile gösterilir.

Noktaları kapsayan hücrelerin hepsi dış bükey poligonlardır.

Basit bir örnek olarak iki noktanın Voronoi çizgisini inceleyelim (Bkz. Şekil 2.3.). İki noktayı birleştiren doğru parçasının orta noktasından geçen doğru Voronoi çizgisini verir. Hücrelerden p noktasını kapsayan hücre Vor(p) ve q noktasını kapsayan hücre ise Vor(q) ile ifade edilir. Üçüncü nokta eklendiğinde üç noktayı birleştiren doğru parçalarının orta noktasından geçen doğruların kesişimi ile orta noktalara çizilen doğrularla üç noktanın Voronoi çizgesi elde edilir (Bkz. Şekil 2.4.).

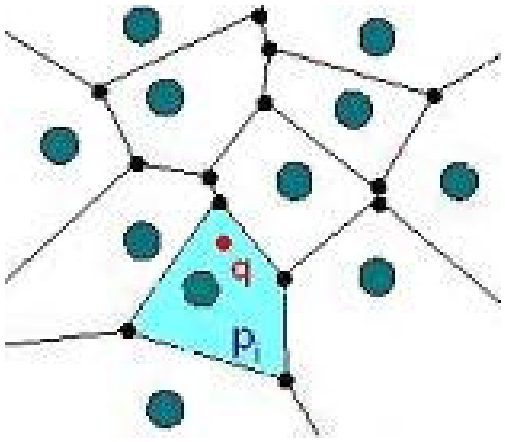



Şekil 2.3. İki Noktanın Voronoi Çizgesi



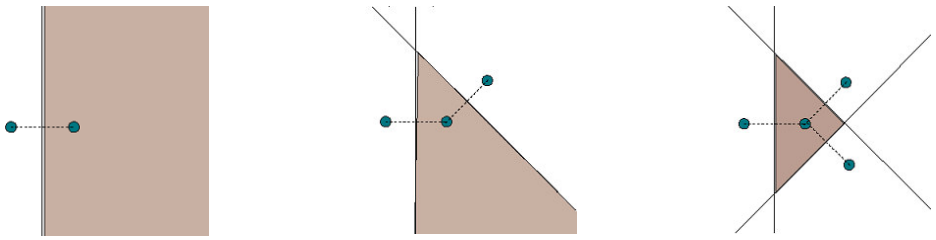

Şekil 2.4. Üç Noktanın Voronoi Çizgesi

Hücrelerin her hangi biri içerisinde alınacak q gibi bir noktaya en yakın nokta, hücreye adını veren noktadır. Başka bir ifade, $Vor(p_i)$ hücresinin iç bölgesinde alınacak her hangi bir q noktasına düzlemde en yakın nokta $p_i$ noktasıdır (Bkz. Şekil 2.5.). (UCSB_Web, sf:5,6) Başka bir ifade ile, her hangi bir q noktası $Vor(p_i)$ hücresi içende olması için gerekli ve yeterli koşul her $p_j \in P$ için $|q-p_i| < |q-p_j|$ ve $i \neq j$ olmasıdır. (Kreveld, sf:147) (Miu)




Şekil 2.5. Voronoi Çizgeleri

Voronoi hücrelerini daha ayrıntı inceleyelim. Düzlemdeki p ve q gibi iki noktayı birleştiren doğrunun orta noktasından geçen dik doğru bölgeyi iki yarı bölgeye böler (Bkz. Şekil 2.6.a). Bu yarı bölgeleri p noktasını kapsayan bölge için h(p,q) ile ve q noktasını kapsayan bölge için h(q,p) ile gösterelim. Voronoi tanımdaki gibi h(p,q) bölgesinde alınan r gibi bir noktanın p noktasına olan uzaklığı q noktasına olan uzaklığından daha azdır. Üçüncü nokta ile oluşan yarı bölgeler ve kesişimleri Şekil 2.6.b'de görülmektedir. Dört noktanın yarı bölgelerinin kesişimi ise Şekil 2.6.c'de görülmektedir. Dikkat edilirse, h(q,p) ifadesi p noktasına komşu olan q noktasının içinde bulunduğu yarı bölge ifade edilmektedir. İfadeyi genellersek, $h(p_i,p_j)$ ifadesi ile $p_i$ yarı bölgedeki noktayı ve $p_j$ komşu noktaları ifade edilir Her $p_i$ noktasının $p_j$ noktası gibi komşu nokta sayısı kadar yarı bölgesi mevcuttur ve bu bölgelerin kesişimi $p_i$ voronoi hücresini verir (Bkz. Şekil 2.6.c). Özetle, $Vor(p_i) = \bigcap_{1 \le j \le n, j \ne i} h(p_i p_j)$ dir (Mount).



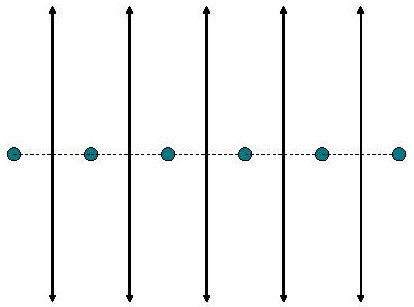
- a - - b - - c -

Şekil 2.6. Voronoi hücre

Görüleceği üzere, n adet noktadan oluşan bir bölgede Vor($p_i$) hücresi için $i \neq j$ olduğu için en fazla ($n - 1$) adet yarı bölgenin kesişiminden oluşabilir.

**Teorem :** P düzlemi n farklı noktadan oluşan bir düzlem olsun. Eğer, düzlemdeki bütün noktalar aynı doğrultu üzerindeyse Vor(P) çizgesi ($n - 1$) adet paralel doğru ve n adet hücreden oluşur. Düzlemdeki noktalar aynı doğrultu üzerinde değilse Vor(P) bağlantılıdır ve kenar çizgileri doğru parçaları ya da bir ucu sonsuza giden yarı doğrulardan oluşur (Kreveld, sf:147).

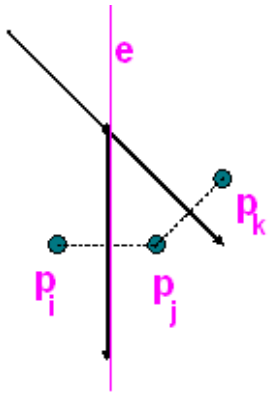
**Kanıt :** Bütün noktalar aynı doğrultu üzerindeyse komşu noktaları birleştiren doğruların orta noktalarından geçen dik doğrular Vor(P)'yi meydana getir. Vor(p)'nin ($n - 1$) adet paralel doğrudan oluşacağı aşikardır (Bkz. Şekil 2.7.).



Şekil 2.7. Aynı doğrultudaki noktaların Voronoi Çizgeleri

Varsayalım ki, bütün noktalar aynı doğrultu üzerinde olmasın. Önce kenar çizgilerinin doğru parçaları ya da bir ucu sonsuza giden yarı doğrulardan oluştuğunu gösterelim. P düzlemini kesen her iki ucu sonsuza giden bir doğru alalım ve bu e doğrusu Vor($p_i$) ve Vor($p_j$) Voronoi hücrelerinin ortak kenar çizgisi üzerinden geçsin (Bkz. Şekil 2.8.). Ayrıca, düzlemdeki noktalardan $p_i$ ve $p_j$ ile aynı doğrultu da olmayan komşu noktalardan bir nokta $p_k$ alalım. $p_j$ ile $p_k$ noktalarını birleştirecek doğru parçasının ortasından ve doğru parçasına dik geçen doğru e doğrusuna paralel olamayacağından e doğrusu ile bir noktada kesişecektir. Kesişim noktası e doğrusunu iki yarı doğruya böler ve bu parçalardan biri $h(p_k,p_j)$ yarı bölgesi içerisinde, diğeri ise $h(p_j,p_k)$ yarı bölgesi içerisinde kalacaktır. $h(p_k,p_j)$ yarı bölgesinde kalan doğru Vor($p_j$) hücresinin sınır çizgisi olamayacağı anlamına gelmektedir (Bkz. Şekil 2.8.). Yani, e doğrunun tamamı değil, sadece $h(p_j,p_k)$ yarı

bölgesinde kalan e yarı doğrusu hücrenin kenar çizgisi olabilmektedir. Eğer, e yarı doğrusunu kesecek şekilde ikinci bir yarı doğru yoksa e doğrusunun tamamı Voronoi hücrenin kenar çizgisi olabilecektir. Eğer, e yarı doğrusunu kesecek şekilde $p_m$ gibi dördüncü bir nokta ile $p_j$ noktalarını ayıran ikinci bir yarı doğru varsa e doğrusuyla arada kalan doğru parçası $p_j$'nin bir kenarını oluşturacaktır.



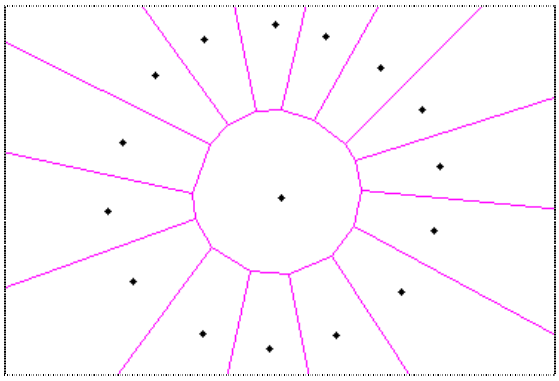

Şekil 2.8. Voronoi hücre

Vor(P) bağlantılı olduğunu gösterirsek kanıt biter. Vor($p_i$)'lerin birleşimi Vor(P)'yi oluşturmaktadır ve Vor($p_i$) hücreleri dış bükey poligonlardır. Eğer, Vor($p_i$) iki paralel kenardan oluşsa idi sorun olabilirdi. Fakat, biz aynı doğrultuda olmayan noktaların yarı doğrularının birbirini kestiğini ve bu durumda tam doğrunun Voronoi hücrelerinin kenar çizgisi olamayacağını gösterdik. ♦

Voronoi çizgelerini komşu noktaları birleştiren doğru parçasına dik ve doğrunun orta noktasından geçen doğru olarak belirmiştik. Şekil 2.6.-c'de görüleceği üzere, hücreye ait noktanın sadece üç adet komşu noktası vardır. Bu noktaları birleştiren doğrulara dik ve orta noktalarından geçen çizgilerin kesişimi ile hücrenin kenar çizgileri oluşur. Kesişim noktaları ise Voronoi hücresinin Voronoi köşelerini oluşturur. Düzlemdeki n adet noktanın Şekil 2.9'deki gibi düzlemde bulunduğunu varsayalım. Görüleceği üzere, orta bölgedeki nokta gibi bir noktanın çevresi $(n - 1)$ adet nokta çevrili olabilir. Bu durumda noktaları birleştiren doğrulara alınacak orta dik doğruların sayısı en fazla $(n - 1)$ adet olabilir.

**Teorem :** Düzlemde bulunan n adet noktanın Voronoi çizgelerinin kenar sayısı en fazla (3n-6) ve köşe sayısı en fazla (2n-5) olabilir (Kreveld, sf:146).

**Kanıt** : Eğer düzlemdeki noktalar aynı doğrultu üzerinde ise; n adet noktanın Voronoi çizgeleri birbirlerine paralel olacak şekildedir ve kenar sayısı (n – 1) adettir ve köşe mevcut değildir (Bkz. Şekil 2.7.).Dolayısıyla, n>3 olmak şartı ile (n - 1) ≤ (3n - 6) olacağı açıktır.



Şekil 2.9. Voronoi Çizgeleri

Varsayalım ki, düzlemdeki noktalar aynı doğrultu üzerinde olmasın. Kanıt için Euler formülünü kullanacağız. Yüzey sayısı $m_f$, eğri sayısı $m_e$ ve nokta sayısı $m_v$ olan bağlantı düzlemsel çizgesi için Euler formülü ile $m_v - m_e + m_f = 2$ elde edilir. Bu formülü Voronoi çizgelerine direkt uygulayamayız. Çünkü, Voronoi çizgelerinde doğru parçalarının yanında ucu sonsuza giden yarı doğrular da mevcuttur. Bu durumu ortadan kaldırmak için düzleme ekstra bir Voronoi köşe ekleyip ucu sonsuza giden yarı doğruların sonsuza giden uçlarını eklenen köşeye bağlanır. Bu durumda Voronoi çizgeleri bağlantılı düzlemsel çizge özelliğine kavuşacak olup artık Euler formülünü kullanabiliriz. Vor(P)'deki köşe sayısına v ve kenar sayısına e diyelim. Düzleme eklenen ekstra köşe ile köşe sayısı v + 1 olacağına da dikkat edersek Euler formülünü $(v + 1) - e + n = 2$ olacak şekilde uyarlayabiliriz. Köşeye değen doğru parçasının sayısını ifade etmek için deg(v) ifadesini kullanırsak her köşe en az üç kenarın kesişiminden oluştuğu için $\deg(v) \geq 3$ tür. Ayrıca, her kenar iki adet köşeden oluştuğu için bütün köşelerin derecelerinin toplamı kenar sayısının iki katı kadardır. Yani,

$\sum_{v \in Vor(p)} \deg(v) = 2e$ dir. deg(v)≥3 olduğu için 2e ≥ 3(v+1) elde edilir.

(v+1) – e + n = 2 den elde edilen (v + 1) = e - n + 2 eşitsizlik $2e \geq 3(v+1)$ de yerine konulursa $e \leq 3n-6$ elde edilir. Aynı şekilde, (v + 1) – e + n = 2 den elde edilen e=(v +1) + n – 2 eşitsizlik $2e \geq 3(v+1)$ de yerine konulursa $v \leq 2n-5$ elde edilir.♦

**Teorem :** Düzlemdeki noktalar kümesi P olsun. P kümesinin oluşturduğu Voronoi çizgesini Vor(P) ile gösterelim.

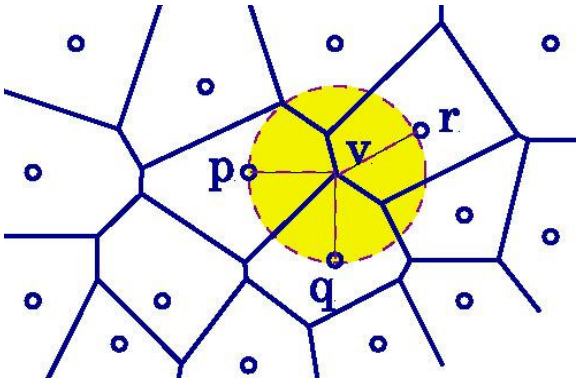
i) Vor(P) çizgesinde alınan bir v noktasının köşe nokta olması için gerek ve yeter şart v merkezli çizilebilecek en büyük boş çember P kümesine ait noktalarından en az üç ya da daha fazla sayıda noktası üzerinden geçmesidir.

ii) Düzlemdeki $p_i$ ve $p_j$ gibi iki komşu noktayı birleştiren doğrunun orta noktasına çizilen dik doğru Vor(P)'de bir kenar çizgisini tanımlaması için gerek ve yeter şart evrensel kümede alınacak q gibi bir nokta ile q merkezli çemberin sadece $p_i$ ve $p_j$ noktalarında geçmesi ve çemberin iç bölgesinde P kümesine ait hiç bir nokta bulunmasıdır.

**Kanıt :** i) Varsayalım ki, Vor(P) bölgesinde alınacak v gibi bir nokta merkezli çizilecek bir çember P kümesine ait en az üç noktadan geçsin. Bu noktalar p, q ve r noktaları olsun. Her bir Vor(p), Vor(q) ve Vor(r) hücresinin sadece kendisine adını veren noktayı içerdiğini, başka bir nokta içermediğini belirtmiştik. Dolayısıyla v noktası yalnızca hücrelerin sınır çizgisi üzerinde olabilir. Üç adet sınır çizgisi paralel olmadığı için v noktası üç noktanın kesişimindedir. Bu durum v noktasının Vor(P)'de bir köşe olduğunu gösterir (Bkz. Şekil 2.10.). Varsayalım ki, Vor(P) bölgesinde alınan v noktası Vor(P)'de bir köşe nokta olsun. Voronoi çizgelerinde her bir köşe en az üç adet kenar doğrusunun kesişiminden oluşmaktadır. Dolayısıyla kenar çizgilerine sahip olan Vor(p), Vor(q) ve Vor(r) gibi en az üç adet Voronoi hücreleri mevcuttur. Kenar çizgisi üzerinde alınacak v gibi bir nokta hücrelere adını veren p, q ve r noktalarına eşit uzaklıktadır (Bkz. Şekil 2.10.). Dolayısıyla, v merkezli çizilecek bir çember p, q ve r noktaları olmak üzere en az üç noktadan geçecektir.

ii) Varsayalım ki düzlemde alınacak her hangi bir q noktası hipotezi sağlayan bir nokta olsun. Yani, q nokta merkezli çizilecek çemberin iç bölgesinde hiç bir nokta bulunmasın ve çember sadece $p_i$ ve $p_j$ noktalarında geçsin. Bu

durumda q noktasının $p_i$ ve $p_j$ noktalarına olan uzaklığı eşit mesafededir. Ayrıca, P kümesinden alınacak her hangi bir $p_k$ gibi bir noktanın q noktasına olan uzaklığı $p_i$ ve $p_j$ noktalarının q noktasına olan uzaklığından büyük ya da eşittir. Eşit olması q merkezli çemberin $p_k$ noktasını da kestiğini gösterir ki, varsayımda çemberin sadece $p_i$ ve $p_j$ noktalarını kestiğini kabul etmiştir. Çember üç noktayı aynı anda kesmediğine göre q noktası Vor(P)'de köşe olamaz. Bu durumda, q noktası Vor(P) hücrelerinin iç bölgesinde olamayacağı ve aynı zamanda köşe de olamayacağına göre q notası Vor(P)'nin sınır çizgileri üzerindedir. Sınır çizgisi üzerindeki q noktasının $p_i$ ve $p_j$ noktalarına olan eşit mesafede olması, sınır çizgisinin $p_i$ ve $p_j$ noktalarının orta dikmesi olduğunu gösterir, yani iki noktayı birleştiren doğru parçasının orta noktasından geçen dik doğru parçasıdır. Yani, kenar çizgisidir. Varsayalım ki, P'de alınan $p_i$ ve $p_j$ gibi iki komşu noktayı birleştiren doğrunun orta noktasına çizilen dik doğru Vor(P)'de bir kenar çizgisi olsun. Kenar çizgisi üzerinde alınacak q gibi bir nokta $p_i$ ve $p_j$ noktalarına eşit uzaklıkta olacaktır ki, q merkezli çizilecek çember $p_i$ ve $p_j$ noktalarından geçer. q noktasına en yakın noktalar $p_i$ ve $p_j$ noktaları olduğu için daha yakın mesafede üçüncü bir nokta mevcut değildir. Dolayısı ile q merkezli çizilecek çemberin üstünde ve içerisinde üçüncü bir nokta mevcut değildir. ♦

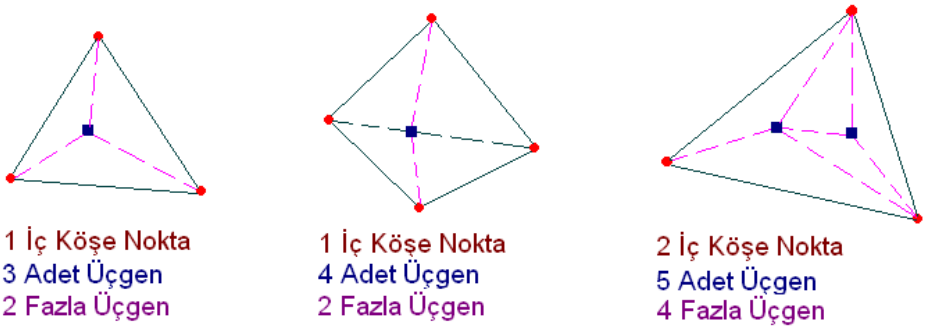



Şekil 2.10. Voronoi Çizgeleri ve üç noktadan geçen çevrimsel çember

**Teorem :** Düzlemdeki hepsi aynı doğrultuda olmayan n adet noktadan oluşan P noktalar kümesinin dış bükey kabuğu içerisinde kalan noktalarının sayısı i olsun. P kümesinin üçgenleştirilmesinden $(n+i-2)$ adet üçgen elde edilir ($n \geq 3$ ve $n, i \in Z$).

**Kanıt :** Düzlemdeki P noktalar kümesinin dış bükey kabuğunda olmayan, yani dış bükey poligonun iç bölgesinde kalan noktalarının sayısına i dersek dış bükey kabuk üzerindeki noktalarının sayısı (`n - i`) adettir. Dış bükey poligonun iç bölgesinde kalan noktalar dikkate alınmayarak sadece dış bükey kabuk üzerindeki noktaların üçgenleştirilmesinden (`(n - i) - 2`) adet üçgene elde edilir. Üçgenleştirilmiş bölgeye eklenecek noktalar ile yapılacak yeni üçgenleştirme ile elde edilecek üçgenlerin sayısı eklenen nokta sayının iki katı oranında daha fazla üçgen elde edilmesini sağlar. (Bkz. Şekil 2.11.). Dolayısıyla iç bölgeye eklenen i adet nokta için 2i adet fazla üçgen elde edilir. Dış bükey üçgen sayısı ile iç noktalardan dolayı oluşan fazla üçgenleri toplamı üçgen sayısını verecektir. Yani, düzlemin üçgenleştirme sayısı

`(n-i-2) + (2i) = n+i-2` ile verilir. ♦



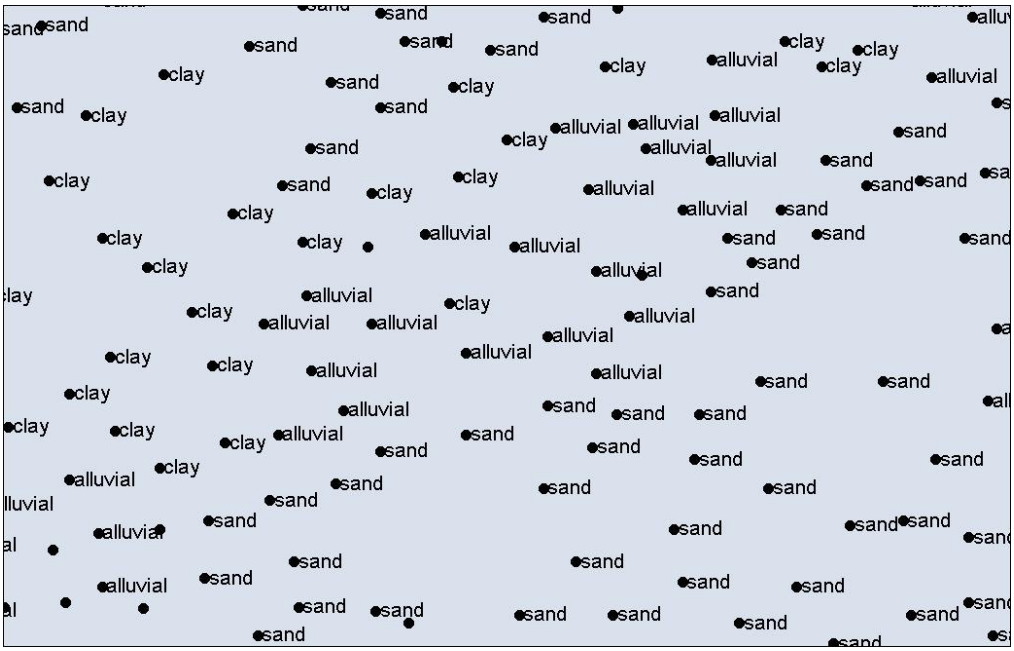

Şekil 2.11. Üçgenin iç bölgesine nokta ekleme

Şu ana kadar Voronoi çizgeleri ve üçgenleştirme ile ilgili bazı teoremleri inceledik. Şimdi Voronoi çizgelerinin güncel hayat dönük kullanımını bir örnek ile inceleyelim.

## 2.1. Voronoi Çizgelerinin Uygulama Alanı

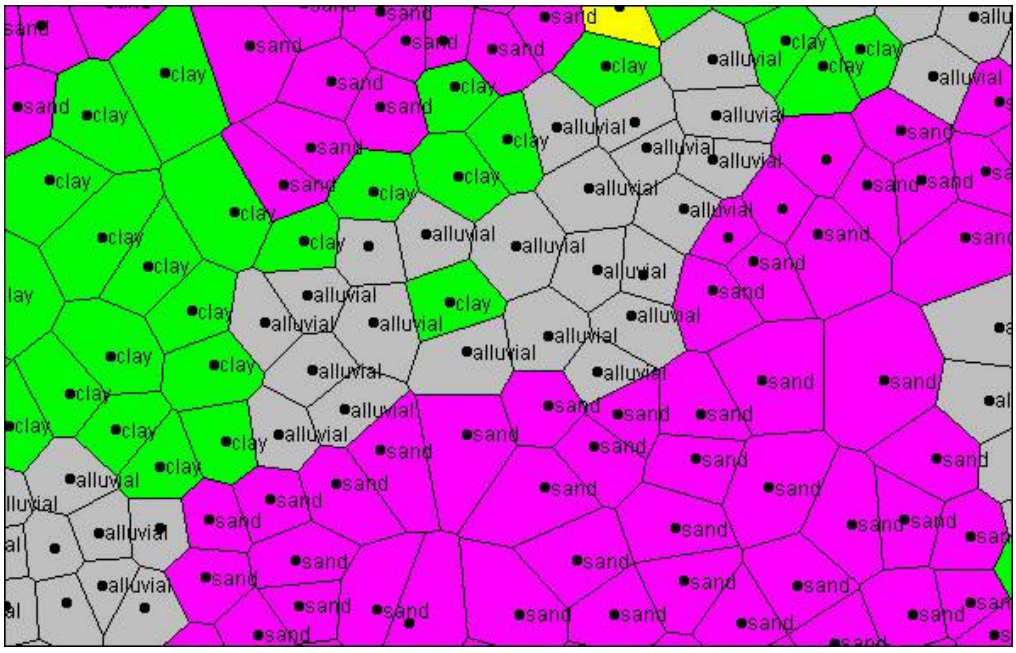
Gerçek hayattaki kullanım alanı genel olarak analizlerdir. Toprak analizi, kirlilik analizi, nüfus yoğunluğu gibi analizler örnek olarak verilebilir.

Arazi üzerinde değişik noktalardan toprak örnekleri (kum/sand, kil/clay, alüvyonlu/humuslu/aluvial toprak) alınmış olsun. Örneklerin alındığı koordinatlar bölge üzerinde işaretlensin (Bkz. Şekil 2.12.).



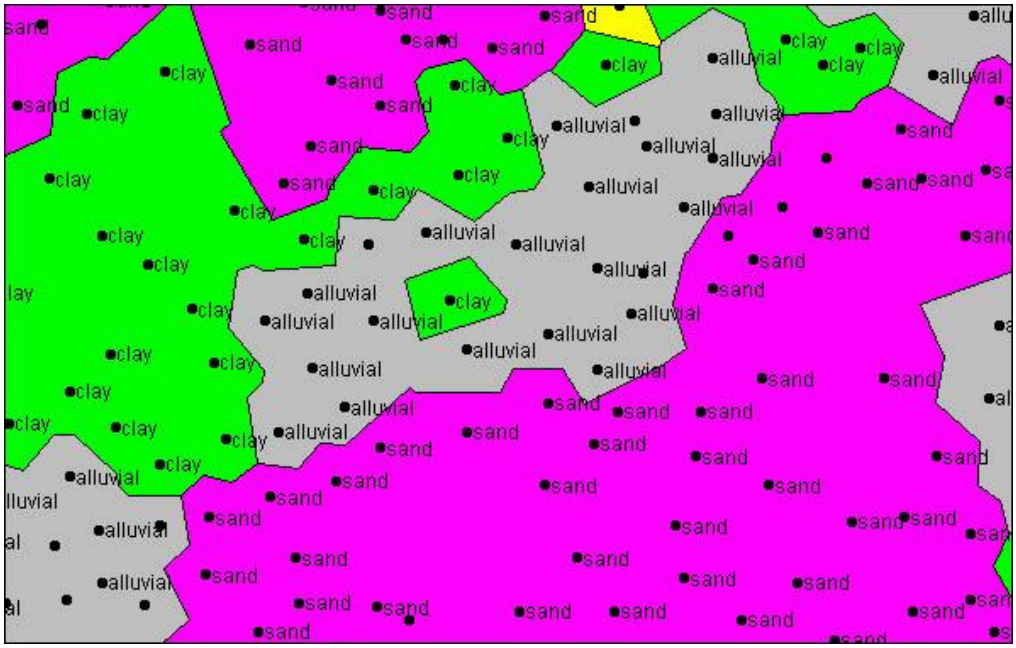

Şekil 2.12. Toprak analizi alınmış bölge

Örneklerin alındığı noktalara göre Voronoi çizgeleri oluşturulup hücreler boyanacak olursa (Bkz. Şekil 2.13.).




Şekil 2.13. Arazi üzerinde Voronoi çizgelerinin oluşturulması

Aynı renkteki hücreler boyanarak birleştirilirse modellenen toprak yapısı elde edilmiş olur (Bkz. Şekil 2.14.). Oluşan modele göre ekim alanları belirlenebilir




Şekil 2.14. Arazi üzerinde verimlilik bölgeleri oluşturma

Voronoi çizgelerinin özelliklerini ve uygulama alanını gördük. Şimdi, Voronoi çizgeleri ve onla bağlantılı olarak Delaunay üçgenleştirmesi aşamalarını inceleyelim.

Düzlemde bulunan nokta sayı n olsun. Noktalara damga numarası vererek birden n'e kadar damgalayalım. Damgalanmış noktalar kümesi için aşağıdaki aşamalar uygulanır.

## 2.2. Delaunay Üçgenleştirme Aşamaları

Üçgenleştirilecek bölgedeki nokta sayısı n olsun. Nokta sayısına göre aşağıdaki işlemler uygulanır.

**I.** n = 1 için; hiçbir işlem yapılmaz.

**II.** n = 2 için;

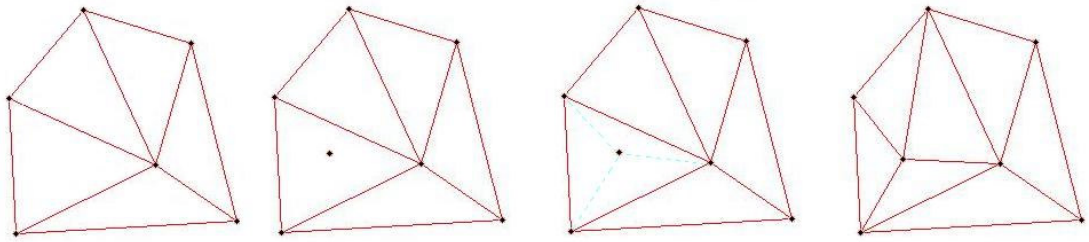
İki noktayı birleştiren doğru parçasının orta dikmesi Voronoi çizgesini verir (Bkz. Şekil 2.3.).

**III.** n = 3 için;

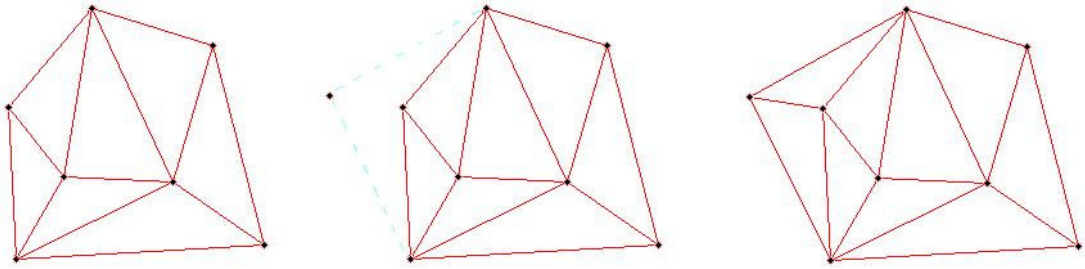
Üç noktanın üç doğru parçası ile birleştirilmesi ile bir adet üçgen elde edilir (Bkz. Şekil 2.4.). Üç noktayı birleştiren üç doğrunun alınacak orta dikmelerinin kesişim noktası Voronoi köşesini verir. Ayrıca, üç noktadan geçen çevrimsel çemberin merkezi de Voronoi köşesini verir. Bulunan köşe noktasından üç noktayı birleştiren üç doğru parçasının orta noktasında geçecek şekilde çizilecek dik doğrular Voronoi çizgelerini verir (Bkz. Şekil 2.4.).

**IV.** n ≥ 4 için;

Damga sırasına göre ilk üç nokta alınır ve bir üçgen çizilir. Sonra, damga sırasına göre yeni bir nokta alınır ve üçgenleştirilen bölgeye eklenir. Eklenen yeni nokta ya üçgen bölge içerisindedir ya da üçgenlerin dışında bir bölgededir. Eklenen nokta iç bölgede ise yeni eklenen nokta ile üçgenin üç köşesine çizilen üç doğru ile noktanın iç bölgesine eklendiği üçgen üç adet üçgene bölünmüş olur (Bkz. Şekil 2.15.) (Bourke). Eklenen nokta dış bölgede ise yeni eklenen nokta ile üçgen bölgenin sınır çizgilerini oluşturan kenarlarından görüş alanında kalan kenar köşe noktaları ile yeni eklenen nokta çizgilerle birleştirilir (Bkz. Şekil 2.16.). Üçgenleştirilmiş bölgenin dış bükey kabuk üzerinde kalan sınır çizgilerinden eklenen noktanın görebildiği kenar adeti kadar üçgen eklenmiş olur. Voronoi çizgeleri kenar orta dikmelerinin kesişimleri ile Voronoi köşeler bulunur ve Voronoi köşeler ile kenar orta noktalarından geçecek şekilde doğrular çizilerek Voronoi çizgeleri çizilir. Süreç bütün noktalar kullanılacak şekilde devam ettirilir. Bu arada, çevrimsel çember içerisinde başka noktalar kalmaması gerekmektedir. Çevrimsel çemberlerin iç bölgelerinde başka noktaların kalması gibi bir durumda Voronoi çizgelerinin özelliğini bozacağından üçgenleştirmede hata yapılmış olur. Hatanın giderilmesi için üçgenler tekrar düzenlenmelidir. Hatalı üçgenlerin düzenlenmesi için kenar dönüşümü işlemi uygulanır. Kenar dönüşümünü ise şu şekilde tanımlayabiliriz; birer kenar ve ikişer köşeleri ortak iki üçgenin ortak kenarı kaldırılıp ortak olmayan köşeleri bir köşegen yardımıyla birleştirilerek yeni iki üçgen elde edilmesi işlemine kenar dönüşümü denir (Bkz. Şekil 2.20.).



Şekil 2.15. İç bölgeye nokta ekleme



Şekil 2.16. Dış bölgeye nokta ekleme

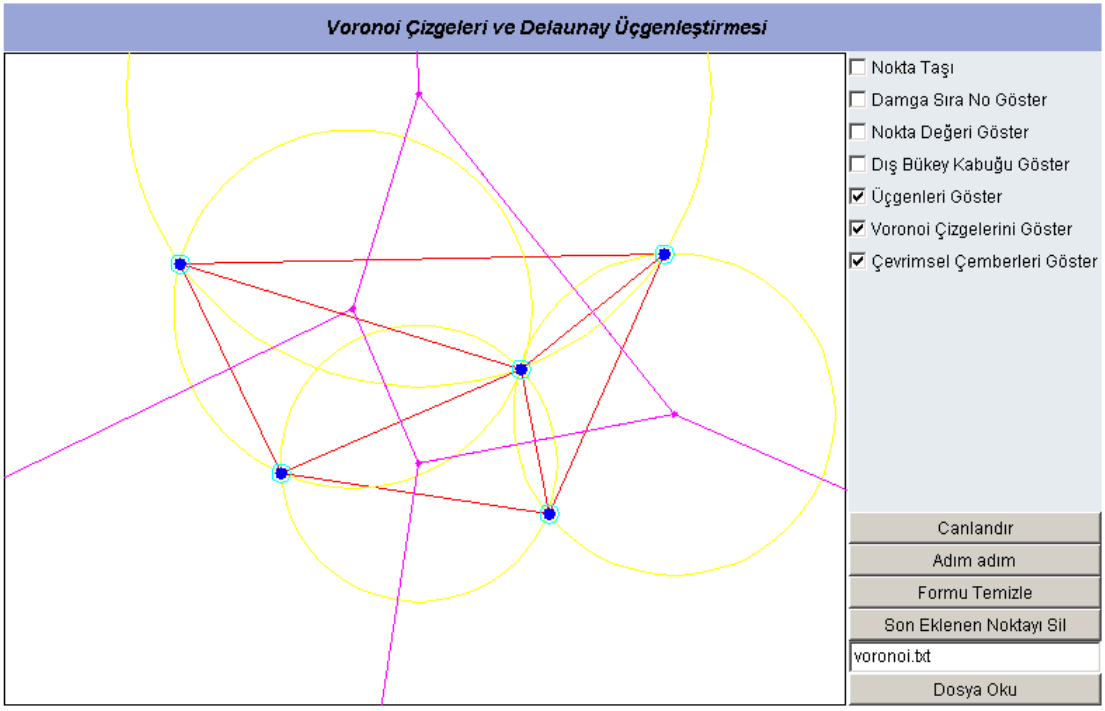
Şimdi, Delaunay üçgenleştirmesindeki geçersiz işlemleri ve nasıl düzeltileceğini inceleyelim. Bu çerçevede kenar dönüşüm yöntemlerini de inceleyeceğiz.

## 2.3. Delaunay Üçgenleştirmesi Ağ Uygulaması

Hazırlanmış olan bu ağ uygulamasında öncelikler üçgenlerin köşe noktalarının girişinin yapılması gerekmektedir. Her eklenen nokta ile delaunay üçgenleştir algoritması çalıştırılır ve üçgenleştirmeler sonucu elde edilen üçgenler çizdirilir. “Üçgen” seçeneği ile üçgenler, “Voronoi Ç.” seçeneği ile voronoi çizgeleri çizdirilir (Bkz. Şekil 2.17.). “Ç.Çember” seçeneği seçilerek üçgenlerin üç köşe noktasından geçen çevrimsel çemberler çizdirilir.

Ayrıca, algoritmanın çalışma mantığı daha kolay anlaşılabilmesi amacıyla ağ uygulamasına “Adım adım” tuşu eklenmiştir. Bu tuşu ile algoritmanın çalışma aşamaları adım adım görselleştirilmektedir. “Adım adım” tuşuna basıldıkça bir sonraki adım görüntülenmektedir. “Canlandırma” tuşu ise “Adım adım” tuşu gibi çalışmakta olup “Canlandırma” tuşu ile her bir adımda görüntü belli bir süre

durdurulup bir sonraki adıma geçilir. Süreç son adıma gelinceye ve üçgenleştirme tamamlanacağı ana kadar devam ettirilir.




Şekil 2.17. Delaunay Üçgenleştirmesi ağ uygulaması

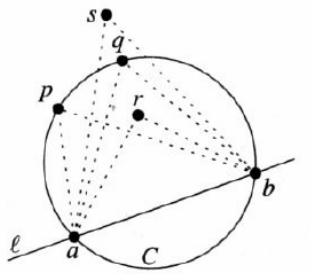
## 2.4. Hatalı Üçgenleştirmeler ve Kontrolü

Üçgenleştirme sonucunda Voronoi köşelerden çizilecek çevrimsel çemberlerin iç bölgesinde düzlemdeki noktalardan her hangi birinin kalması gibi bir durum Voronoi çizgelerinin özelliğini bozacağından üçgenleştirmede hata yapılmış olur. Üçgenleştirmede hata varsa hatalı üçgenler için kenar dönüşümü yapılır. Üzerinde kenar dönüşümü yapılacak kenara geçersiz kenar da denmektedir.

Hata tespitinin nasıl yapılacağını incelemeden öncelikle düzlemdeki noktalar ile çevrimsel çemberler arasındaki açı ilişkisini inceleyelim.

Her hangi bir çember ve çemberi kesen l doğrusunun kesen noktaları a ve b olsun. Çember üzerinde herhangi bir yerde q ve p noktalarını, çember iç bölgesinde r ve çember dış bölgesinde s noktalarını alalım (Bkz. Şekil 2.18.). Çember üzerindeki p ve q noktalarının a ve b noktaları ile yaptığı açılar çember üzerinde aynı yayı gören açılar oldukları için eşittir. İç bölgedeki r noktasının a ve b

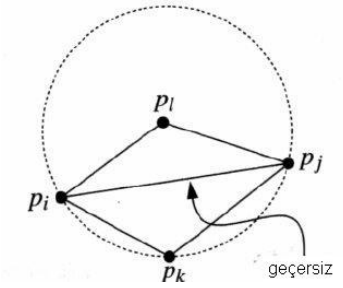
noktaları ile yaptığı açı çember üzerindeki p noktasının a ve b noktaları ile yaptığı açıdan daha büyüktür. Aynı şekilde, dış bölgedeki s noktasının a ve b noktaları ile yaptığı açı çember üzerindeki p noktasının a ve b noktaları ile yaptığı açıdan daha küçüktür. Ayrıca, Şekil 2.19'da olduğu gibi aynı kirişi gören merkez nokta $p_l$'nin açı $p_i$ ve $p_j$ ile yaptığı açı çember üzerindeki $p_k$'nın $p_i$ ve $p_j$ ile yaptığı açıdan daha büyüktür.



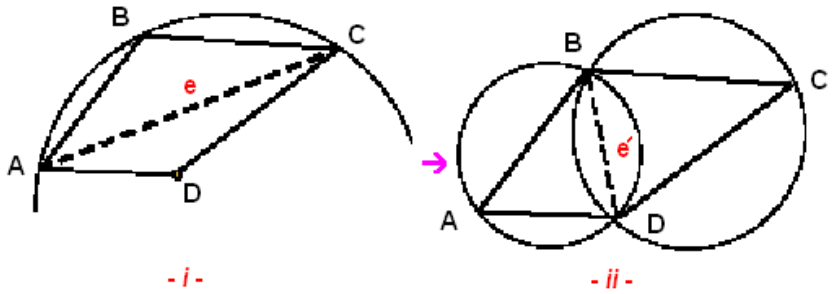

Şekil 2.18. Çember

Yapılan üçgenleştirmede $p_i$, $p_j$, $p_k$ gibi üç noktanın oluşturduğu $p_i$, $p_j$ , $p_k$ üçgenin çevrimsel çemberi içinde $p_l$ gibi bir nokta kalıyor ise üçgen hatalıdır ve $p_ip_j$ kenarı için kenar dönüşümü yapılmalıdır (Bkz. Şekil 2.19.).

Sorunu daha iyi kavrayabilmek için Şekil 2.20'de inceleyelim. Şekil 2.20-i de D köşesinin $T_{ABC}$ üçgeninin çevrimsel çemberi içinde kaldığı görülmektedir. D köşesinin çevrimsel çember içerisinde kalmaması için e kenarı kaldırılır ve e´ kenarı yeni kenar olarak atanırsa $T_{ADB}$ ve $T_{BDC}$ üçgenleri elde edilmiş olur. $T_{ADD}$ ve $T_{BDC}$ üçgenleri ile sorunun giderilmiş olduğu görülecektir (Bkz. Şekil 2.22.i).




Şekil 2.19. Geçersiz kenar




Şekil 2.20. Kenar dönüşümü

Şimdi, kenar dönüşümü nasıl yapılır, inceleyelim.

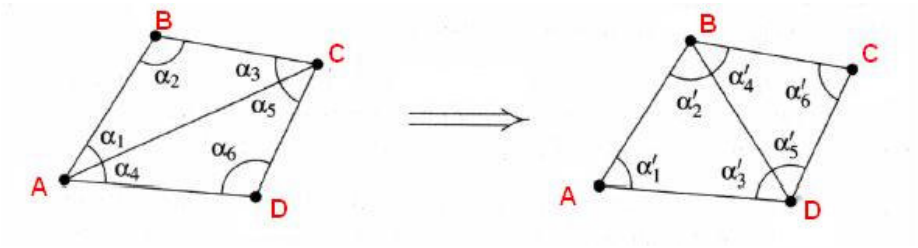
## 2.5. Üçgenleştirmede Kenar Dönüşümü

Üçgenleştirmede hedef üçgenlerin eş kenarlı üçgene en yakın üçgenler olacak şekilde elde edilmesidir. Yani, bütün iç açıların $60^0$'ye yakın olmasına çalışılmasıdır.

Geçersiz kenarın bulunmasında üçgenlerin iç açılarından faydalanılır. Örneğin, düzlemde bulunan dört nokta ile Şekil 2.21'de görüleceği üzere iki farklı üçgenleştirme yapılabilir. Biri [AC] köşegeni ile elde edilen $T_{ACB}$ ve $T_{CAD}$ üçgenleştirmesi, diğeri ise [BD] köşegeni ile $T_{BDA}$ ve $T_{BDC}$ üçgenleştirmesidir. $T_{ACB}$ ve $T_{CAD}$ üçgenlerine ait altı adet iç açının en küçük açı değeri m olsun. $T_{BDA}$ ve $T_{BDC}$ üçgenlerine ait altı adet iç açının en küçük açı değeri ise m´olsun. Yapılan üçgenleşmelerden geçerli olanı bulmak için bulunan iki açı karşılaştırılır. Bulunan değerlerden m < m´ ise [AC] köşegeni geçersiz kenardır, m > m´ise [BD] köşegeni geçersiz kenardır. Başka bir ifadeyle, Şekil 2.21'de üçgenler üzerinde yapılacak iç açı incelemesi sonucunda: (Eguchi)(Ostrovsky)(Çetin)

$\min_{1\le i\le 6} \alpha_i < \min_{1\le i\le 6} \alpha_i' \Rightarrow$ [AC] geçersiz kenardır,

$\min_{1\le i\le 6} \alpha_i > \min_{1\le i\le 6} \alpha_i' \Rightarrow$ [BD] geçersiz kenardır.



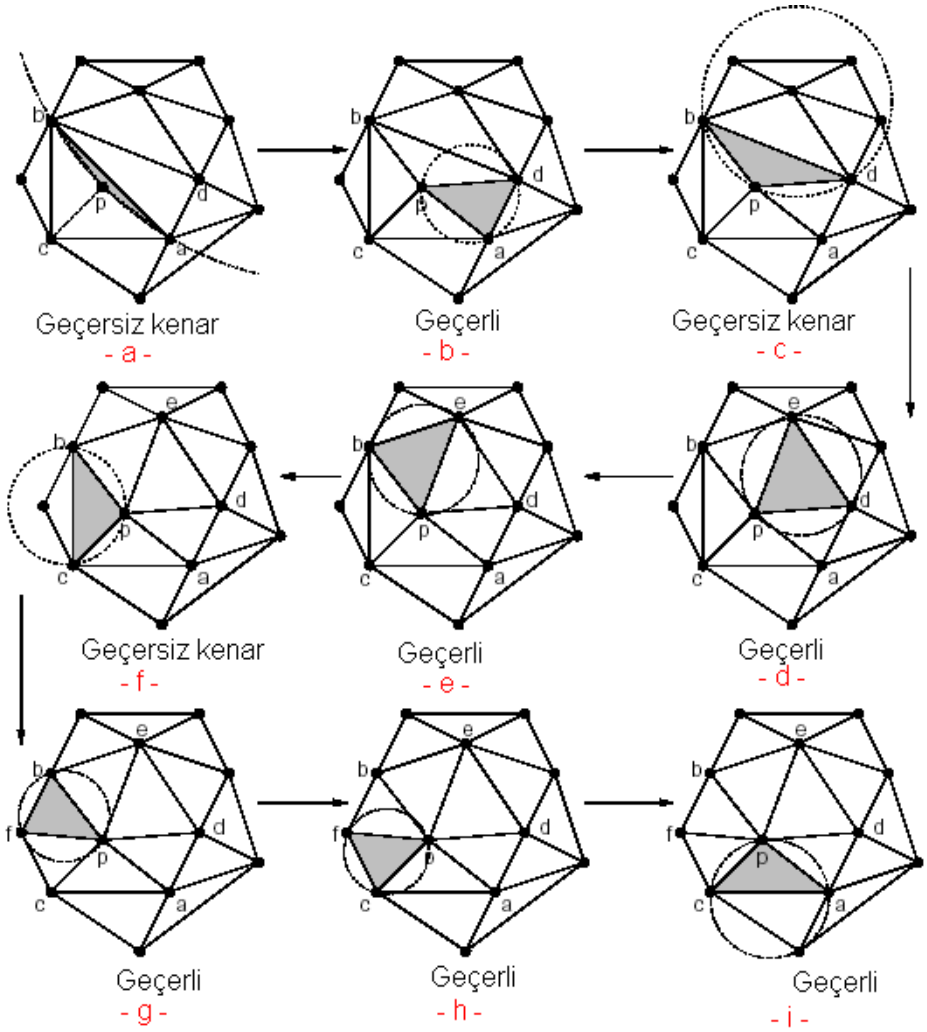

Şekil 2.21. Kenar dönüşümünde iç açıların kullanılması

Üçgenleştirme aşamasında [AC] köşegeni ile $T_{ACB}$ ve $T_{CAD}$ üçgenleştirmesi yapılmış ve yapılan kontrolde [AC] köşegeninin geçersiz olduğu ve kaldırılması gerektiği tespit edilmiş olsun. Şekil 2.21'deki [AC] köşegeni kaldırılmaz, başka bir ifade ile kenar dönüşümü yapılmaz ise Şekil 2.20-i'ye benzer bir şekilde $T_{ABC}$ çevrimsel çemberi içerisinde D noktası kalacaktır.

Kenar dönüşümü için önce [AC] köşegeninin kenarını oluşturduğu, başka bir ifade ile A ve C noktalarını köşe noktası kabul eden iki üçgen bulunur. Bulunan üçgenler geçersiz oldukları için kaldırılacaklardır. İki üçgene ait A ve C köşeleri dışında olan üçüncü köşeleri olan B ve D noktaları tespit edilir. [AC] köşegeni kaldırılıp [BD] köşegeni çizilerek $T_{BDA}$ ve $T_{BDC}$ üçgenleri oluşturulur, kenar dönüşümü tamamlanır.

Dikkat edilmesi gerekli bir durum, bir kenar dönüşümünün yeni kenar dönüşümlerini gerektirip gerektirmediğinin de kontrol edilmesi gerekmektedir (Bkz. Şekil 2.22.). Şekil 2.22.a' da görüleceği üzere a, b, p köşe noktalarından geçen çevrimsel çemberin iç bölgesine d noktası da dahil olmak üzere en az bir adet nokta girmektedir. Bu durum, $T_{abp}$ üçgeninin hatalı olduğunu ve tekrar düzenlenmesi gerektiğini göstermektedir. $T_{abp}$ üçgeni ile $T_{adb}$ üçgeninin ortak kenar olan [ab] kenarı geçersiz kenardır. Başka bir ifade ile, $T_{abp}$ ve $T_{adb}$ üçgenlerinin iç açılarının en küçük değeri x ve $T_{adp}$ ve $T_{dbp}$ üçgenlerinin iç açılarının en küçük açı değere y ise $x < y$ olacaktır. [ab] köşegeni kaldırılarak [pd] köşegeni eklenir. Üçgenleştirmede $T_{abp}$ ve $T_{adb}$ üçgenleri kaldırılmış, $T_{adp}$ ve $T_{dbp}$ üçgenleri yeni üçgenler olacak şekilde eklenmiş olur. Yapılan bu düzenleme yeni bir geçersiz kenar oluşmasına sebep olmuş mu, kontrol edilmeli. Önce, $T_{adp}$ üçgenine ait üç kenar olan [ad], [dp] ve [pa] kenarları üçgenleştirme iç bölgesinde ise mutlaka iki üçgene ait kenarlar olmaları gerekmektedir. Yapılan incelemede sorun olmadığı

anlaşılmış (Bkz. Şekil 2.22.b). Sonra, Önce, $T_{dbp}$ üçgenine ait üç kenar olan [db], [bp] ve [pd] kenarları incelenir. Yapılan inceleme sonucunda tanımlı $T_{dbp}$ ve $T_{bpe}$ üçgenlerine ait iç açılar ile sanal tanımlı $T_{pde}$ ve $T_{peb}$ üçgenlerine ait iç açılar karşılaştırılır. Karşılaştırma sonucunda [bd] kenarının geçersiz olduğu tespit edilir (Bkz. Şekil 2.22.c). Sonuçta, $T_{dbp}$ ve $T_{bpe}$ üçgenleri ve dolayısıyla [bd] köşegeni kenarı kaldırılır, [pe] köşegeni ile $T_{pde}$ ve $T_{peb}$ üçgenleri üçgenleştirmeye dahil edilir. Üçgenleştirmeye eklenen her yeni üçgen, kenarları tek tek incelenerek geçersiz kenar varsa tekrar üçgenleştirilerek süreç devam ettirilir. Yeni eklenen üçgenlerin incelemesi bittiğinde geçersiz kenar kalmayacaktır (Bkz Şekil 2.22.i). (Mount)

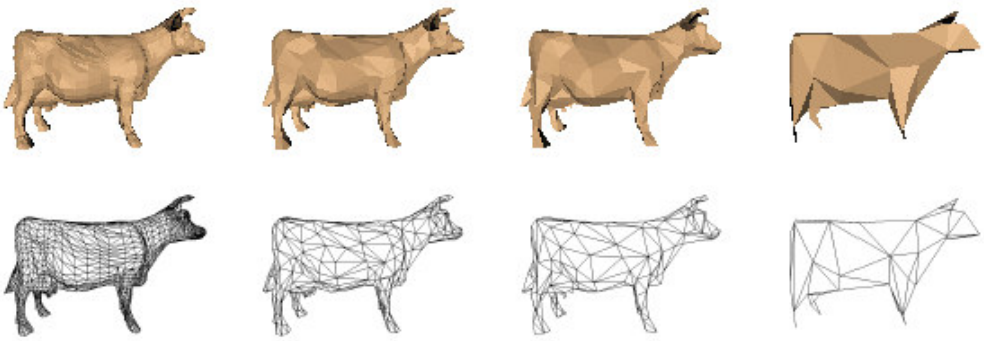



Şekil 2.22. Geçersiz kenarların geçerli hale dönüştürülmesi

## 3. POLİGONAL BASİTLEŞTİRME

Bilgisayar grafiğinde nesneleri ifade etmek için nokta, doğru, poligon ve yay parçaları gibi geometrik ilkeller sık sık kullanılır. Bazen bir nesneyi ifade etmek için çok sayıda poligon kullanılır. Poligon basitleştirilmesi işlemini, kullanılan poligon sayısının nesnenin orijinal görünümü bozulmayacak şekilde azaltılması şeklinde özetleyebiliriz (Bkz. Şekil 3.1.).

Bilgisayarla canlandırmalar hazırlanırken görüntü kalitesine göre saniyede 24 ile 30 adet kare kullanılmaktadır. Her bir karede ise binlerce poligon kullanılmakta ve boyanmaktadır. Poligon sayısı artıkça bilgisayarlara düşen yük o oranda artmakta ve aynı oranda performans düşüklüğü görülmektedir. Poligonal basitleştirme, model karmaşıklığı (poligon sayısı) ile donanım performansı arasındaki bu farklılığı azaltmayı amaçlamaktadır. Böylece, modeller daha çabuk çizilebilecek, değişik gereksinimlere göre modellerin değişik detaylardaki çalışmaları kalite ve zaman kıstasları göz önüne alınarak kullanılabilecektir. (Güdükbay, Sf:1,2)



*(a) 5804 Üçgen* *(b) 1000 Üçgen* *(c) 500 Üçgen* *(d) 100 Üçgen*

Şekil 3.1. Farklı sayıda üçgenleştirilmiş inek nesnesi

Poligon basitleştirmesinin getirdiği en büyük yarar, görüntüleme performansının artmasıdır. Bu, özellikle dinamik ortamlarda görüntü boyama işleminin hızlandırılması açısından önemlidir. Ayrıca, bilgisayar destekli tasarım, reklamcılık ve tıp alanlarında faydalanılabilecektir. Tıp alanında bilgisayarlı tomografi (CT) araçları ile alınan görüntülerden elde edilen üç boyutlu modellerin basitleştirilmesi, bu modellerin hızlı bir şekilde gösterilebilmeleri açısından çok

önemlidir. Ayrıca, basitleştirme işlemi bir modelin saklanması için gerekli bellek miktarını da azaltmaktadır. Böylece bu modellerin bilgisayar ağları üzerinde bir noktadan başka bir noktaya transferi de çabuklaştırılmaktadır. Ayrıca, poligon basitleştirme bilgisayar grafiğinin yoğun hesaplamalar gerektiren (ışın izleme, çarpışma tespiti gibi poligon sayısı ile verimliliği doğrudan ilişkili) problemlerinin çözümlerinde de verimlilik sağlamaktadır. Sonuç olarak bu yöntemlerden iletişim, tıp, bilgisayar destekli tasarım, reklamcılık, canlandırma ve bilimsel gösterim alanlarında yararlanılabilecektir (Güdükbay, Sf:1,2).

Poligon basitleştirme işlemi poligon sayısının azaltması şeklinde de özetlenebilir. Poligon sayısı azaltılırken görüntü kalitesinde düşme olacaktır. Şekil 3.1'de de görüleceği gibi aynı nesne 5804, 1000, 500 ve 100 adet üçgenle de görüntülenebilir. Üçgen sayısı 5804 olan görüntüdeki kalite ile üçgen sayısı 100 olan görüntü kalitesi bir olmadığı, üçgen sayısı azaltıldıkça görüntü kalitesinde de düşme olduğu açıkça görülmektedir (Bkz. Şekil 3.1.d).

Görüntüleme de nesnenin bulunduğu konum da bizim için önemlidir. Görüntülenecek olan nesnenin yakın ya da uzak olması görüntüdeki ayrıntı açısından önemlidir (Bkz. Şekil 3.2.) (Garland2). Göz ile nesne arasındaki mesafe arttıkça nesne üzerindeki ayrıntılar daha az önemli hale gelecektir (Bkz. Şekil 3.3.).

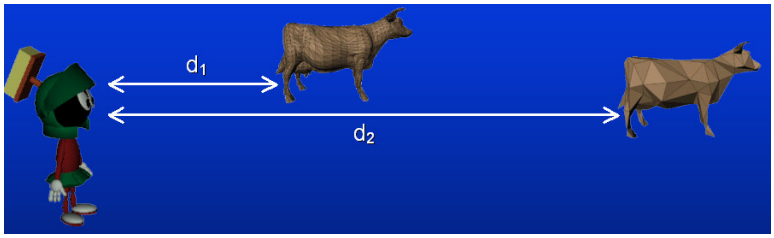
(a) Yakın Görünüm (b) Normal Görünüm (c) Uzak Görünüm

Şekil 3.2. Görüntü kalitesi ile uzaklık ilişkisi

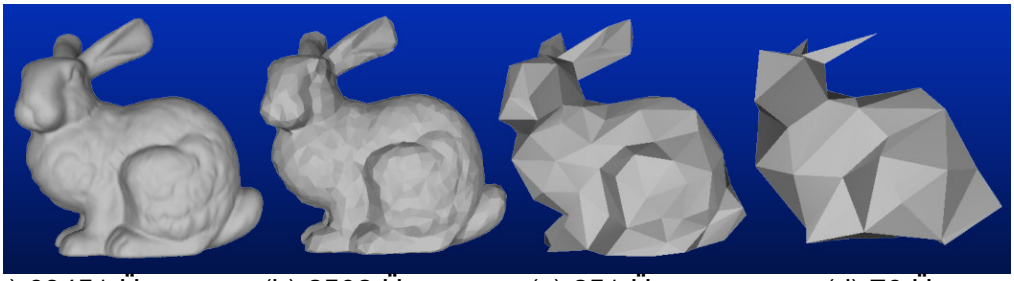
Görüntü kalitesi için poligon sayısı ile nesnenin uzaklık-yakınlık durumunu ilişkilendirebiliriz.

Nesnenin göze olan uzaklığı ile elde edilen görüntü kalitesi arasındaki ilişkiyi daha iyi kavramak için bir örnek daha inceleyelim. Şekil 3.4'de aynı tavşan

nesnesini görüntülemek için değişik sayıda üçgen kullanılmış. Görüntü kalitesi en iyi olan nesnenin 69451 adet üçgenin kullanıldığı nesne olduğu açıktır. Üçgen sayısı azaltıldıkça görüntü kalitesinde bozulma olduğu da gözlemlenmektedir. Dört farklı görüntü kalitesindeki dört adet tavşan nesnesi Şekil 3.5'te üçgen sayısı en fazla olan en yakında olacak şekilde üçgen sayılarıyla orantılı bir şekilde uzaklıkları ayarlanmıştır.

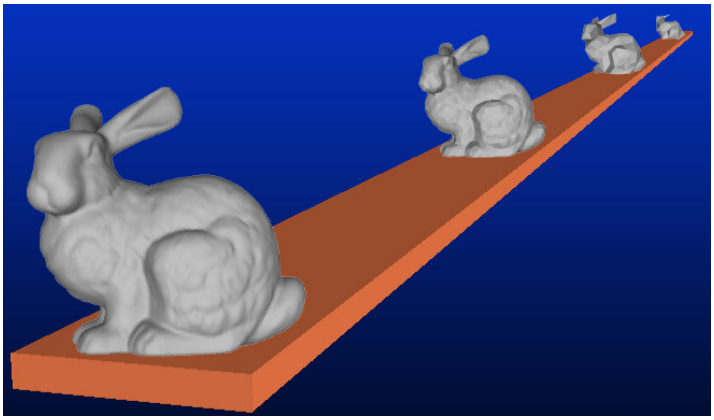



Şekil 3.3. Göz ile nesne arası mesafe



(a) 69451 Üçgen (b) 2502 Üçgen (c) 251 Üçgen (d) 76 Üçgen

Şekil 3.4. Dört farklı görüntü kalitesi



Şekil 3.5. Uzaklık ile görüntü kalitesi arasındaki ilişki

Sonuç olarak, Nesneler uzaklaştıkça üçgen sayısından kaynaklanan görüntü kalitesindeki düşüklük önemsenmeyecek derecede azalmaktadır. Şekil 3.4'deki tavşan nesnelerinin Şekil 3.5'teki uzaklıklarına göre üçgen sayısı ayarlanabilir. Yani, Şekil 3.2.c de olduğu gibi nesne uzak bir görünüm gerektiriyor ise nesneyi fazla sayıda üçgenle ifade etmek anlamsız olacaktır, uzaklıkla orantılı olarak daha az üçgenle de nesne görüntülenebilir.

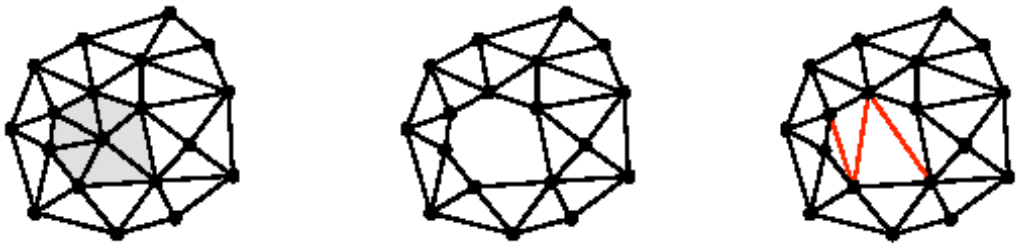
Şimdi, poligonal basitleştirme algoritmalarını ve aralarındaki farklılıkları inceleyelim.

## 3.1. Poligonal Basitleştirme Algoritmaları

Poligonal basitleştirme, iki ya da üç boyutlu poligonal modellerin daha basit bir hale dönüştürülmesi işlemidir. Bu işlem modelin orijinal şekil ve görüntüsüne sadık kalarak modeli tanımlamak için gerekli poligon sayısını azaltır. Poligonal basitleştirme algoritmalarını köşe kaldırma (vertex decimation), kenar büzme (edge contraction), üçgen büzme (triangle collapse), kenar kaldırma, köşelerin birleştirilmesi (vertex clustering) ve örnekleme olmak üzere dört ana başlıkta inceleyebiliriz. (Güdükbay, sf:3)

### 3.1.1. Köşe kaldırma algoritması

Poligonal modelde kaldırılmak üzere bir köşe seçilir ve köşe kaldırılır. Köşenin kaldırılması ile kaldırılan köşeye ait üçgenlerde kaldırılır. Üçgenlerin kalkması ile bir boşluk oluşur (Bkz. Şekil 3.6.). Oluşan boşluk tekrar üçgenleştirilerek boşluk kaldırılır. Bu işlem nesne üzerinde yinelemeli bir şekilde uygulanırsa nesneyi oluşturan üçgen sayısı azaltılmış olacaktır (Güdükbay, sf:3) (HECKBERT, Sf:13).



Şekil 3.6. Köşe kaldırma

**Algoritma:**

Köşe kaldırma algoritması

Girdi: Üçgenler kümesi, köşe noktalar kümesi, kaldırılacak köşe

Çıktı: Yeni üçgenler kümesi, yeni köşe noktalar kümesi

```
KoseKaldırma

  boş bir P´ noktalar kümesi tanımlanır

  while (v köşesini içeren üçgen varsa) {

     üçgenin v köşesi hariç köşelerini P´ kümesine at

     üçgeni üçgenler kümesinden sil

  }  // döngü sonu

  v köşesini gelen noktalar kümesinden sil

  P´ kümesini üçgenleştir

  yeni üçgenler kümesini gelen üçgenler kümesine ekle
```

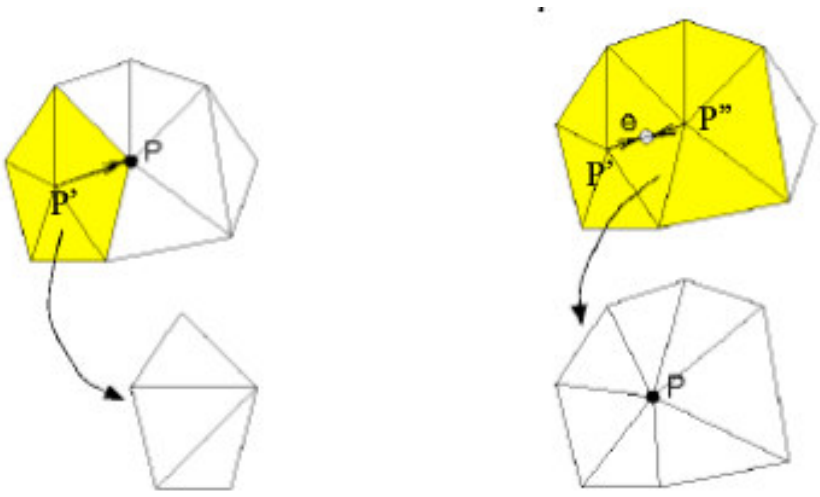
Algoritmanın çalışma hızını artırmak için noktalar kümesinin hata oranına göre artan sırada olması faydalı olacaktır (Puppo).

### 3.1.2. Kenar büzme algoritması

Kaldırılmak üzere seçilen kenar kaldırılır. Kaldırılan kenara sahip olan iki üçgende kaldırılır. Sonra, kaldırılan kenara ait iki uç noktası, sanki bir ip büzülerek iki ucu birleştiriliyormuş gibi bir noktada birleştirilir (Turk). Bu işlem için iki yöntem uygulanır. İki köşeden biri yeni köşe olarak kullanılırsa bu yönteme yarı kenar büzme (Bkz. Şekil 3.7.a), iki köşe arasında uygun yeni bir köşe bulunursa tam kenar büzme denir (Şekil 3.7.b). Elde edilen yeni nokta, kaldırılan noktalara sahip üçgenlerin yeni köşe değeri olacak şekilde atanarak kenar büzme işlemi tamamlanır (Heckbert, sf:17) (Hadwiger) (Luebke, sf:21) (GREİNER, Sf:11).

Bu yöntemi kullanan pek çok algoritma geliştirilmiştir. Bu algoritmaları birbirinden ayıran en önemli özellik, yok edilecek poligon kenarının nasıl seçildiğidir.

Kenar büzme işleminin ters işlemine de köşe bölme denilmektedir. Kenar bölme ile seçilen nokta iki noktaya dönüştürülür. Bu işlem ile iki üçgen eklenmiş olur.




a - Yarı kenar büzme b - Tam kenar büzme

Şekil 3.7. Kenar büzme

### 3.1.2.1. Yarı kenar büzme

Büzülecek kenara ait iki köşeden biri sabit tutularak diğer köşenin sabit tutulan köşeye doğru kaydırılması işlemine yarı kenar büzme denir. Şimdi algoritmanın nasıl çalıştığını inceleyelim.

Büzülecek kenar e kenarı olsun. P ve P' köşeleri e kenarının uç noktaları olsun. P köşesinin P' köşesine doğru kaydırılması ile e kanarını içeren $t_1$ ve $t_2$ üçgenleri büzme işlemiyle birlikte $e_1$ ve $e_2$ kenarlarına dönüşürler (Bkz. Şekil 3.8.). Başka bir ifadeyle $t_1$ ve $t_2$ üçgenleri kaldırılmış olur. Ayrıca, P' köşesini içeren diğer üçgenler büzme işlemi ile P köşesini içerir hale de gelmektedirler.

Algoritmanın işleyişini daha iyi kavrayabilmek için algoritmanın sözde kodlarını inceleyelim (Melax).

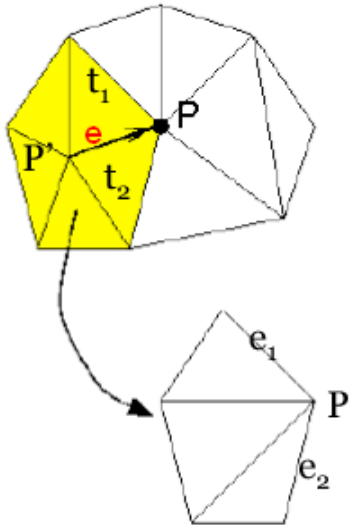
**Algoritma** :

Yarı kenar büzme algoritması

Girdi : Köşe noktalar kümesi, üçgenler kümesi, büzülecek kenar

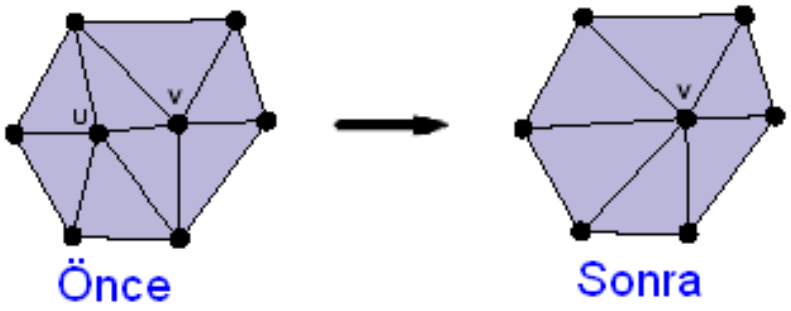
Çıktı : Yeni köşe noktalar kümesi, yeni üçgenler kümesi

```
e kenarını içeren t1 ve t2 üçgenlerini kaldır
while (P' köşesini içeren üçgen varsa)
   P' köşesini P olacak şekilde düzelt
P' köşesinin sil
```




Şekil 3.8. Yarı kenar büzme

Dikkat edilecek olunursa, yarı kenar büzme işlemi uygulama olarak köşe kaldırma işlemine benzemektedir. Büzülecek e kenarına ait iki noktadan v noktası `v noktası üzerine doğru kaydırılır. Bu kaydırma işlemi ile birlikte v köşesini içeren üçgenlerin v köşelerinin değerleri `v olacak şekilde düzenlenir (Bkz. Şekil 3.9.). Aslında, v köşesi kaldırılmış ve bölge tekrar üçgenleştirilmiştir. Köşe kaldırma işleminde de benzer durum söz konusudur (Bkz. Şekil 3.6.) (De Floriani, sf:25).




Şekil 3.9. Yarı kenar büzme

### 3.1.2.2. Tam kenar büzme

Büzülecek kenara ait iki köşe arasında yeni bir köşe bulunarak iki köşenin yeni köşeye doğru kaydırılması işlemine tam kenar büzme denir. Şimdi algoritmanın nasıl çalıştığını inceleyelim

Büzülecek kenar e kenarı olsun. P' ve P" köşeleri e kenarının uç noktaları olsun (Bkz. Şekil 3.10.). P' ve P" köşeleri arasında bulunan yeni köşe Q köşesi olsun. P' ve P" köşeleri Q köşesine doğru kaydırıldığında $t_1$ ve $t_2$ üçgenleri kaldırılmış olur. Ayrıca, P' ve P" köşelerini içeren diğer üçgenler büzme işlemi ile Q köşesini içerir hale geleceklerdir.

Şimdi, kenar büzme algoritmalarından tam kenar büzme algoritmasının sözde kodlarını inceleyelim (Bkz. Şekil 3.10.) (Garland, sf.47) (Hoppe).

**Algoritma** :

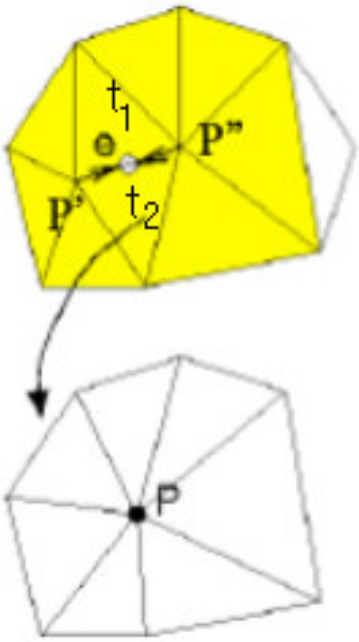
Tam kenar büzme algoritması

Girdi : Köşe noktalar kümesi, üçgenler kümesi, büzülecek kenar

Çıktı : Yeni köşe noktalar kümesi, yeni üçgenler kümesi

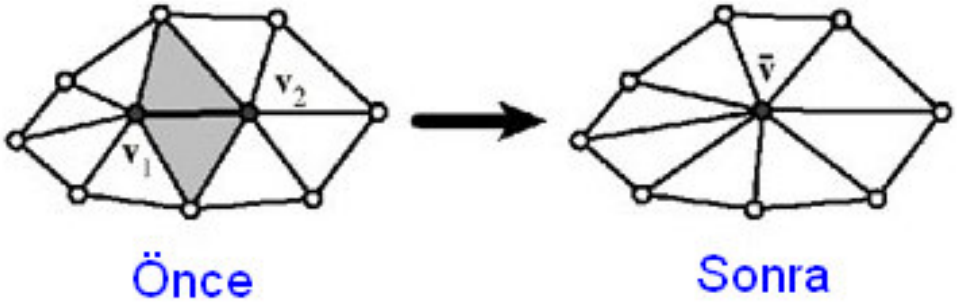
```
e kenarını içeren t1 ve t2 üçgenlerini kaldır
e kenarı üzerinde uygun bir Q noktası bulunur
while (P' köşesini içeren üçgen varsa)
   P' köşesini Q olacak şekilde düzelt
```

```
P’ köşesinin sil

while (P” köşesini içeren üçgen varsa)

   P” köşesini Q olacak şekilde düzelt

P” köşesinin sil
```




Şekil 3.10. Tam kenar büzme

Tam kenar büzme işlemiyle büzülecek kenara ait köşeleri içeren bütün üçgenlerin şekilleri de değişmektedir. (Bkz. Şekil 3.11.)




Şekil 3.11. Tam kenar büzme

### 3.1.3. Köşe çifti büzme algoritması

Mantık olarak kenar büzme işlemine benzer. Kenar büzme işleminde iki köşe yeni köşeye dönüştürülerek iki köşeyi birleştiren kenar ve kenarı içeren iki üçgenin kaldırıldığını belirtmiştir (Bkz. Şekil 3.7.). Fakat, birbirlerine çok yakın köşeler aralarında bir poligon kenarı olmasa dahi birleştirilebilir ki bu yönteme köşe çifti büzme denir. Böylece birbirlerine bağlı olmayan bölgeler birleştirilmiş olacaktır. Bu işlem modelin topolojisini değiştirmekte ancak bu durum çok çözünürlüklü çizim için gerek duyulan bir özellik olmaktadır (Güdükbay, sf:3).

Köşe çifti büzme algoritmasında iki köşeyi birleştiren kenar olmadığı için kenar kaldırma işlemi olmadığı gibi üçgen kaldırma işlemi de yapılamaz (Bkz. Şekil 3.12.). Sadece, iki noktayı içeren üçgenler yeni noktaya göre tekrar güncellenir. Kenar büzme işlemine benzemesi, fakat arada kenar olmaması nedeni ile köşe çifti büzme işlemine hayali kenar büzme de denir (Luebke, sf:23) (Varshney, sf:11).

Şimdi köşe çifti büzme algoritmasının sözde kodlarını inceleyelim (Bkz. Şekil 3.12.).

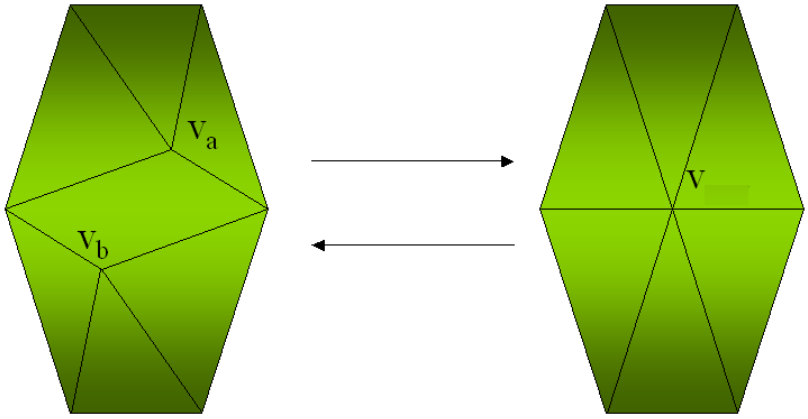
**Algoritma** :

Köşe çifti büzme algoritması

Girdi : Köşe noktalar kümesi, üçgenler kümesi, büzülecek iki köşe

Çıktı : Yeni köşe noktalar kümesi, yeni üçgenler kümesi

```
while (Va köşesini içeren üçgen varsa)
   Va köşesini V olacak şekilde düzelt
Va köşesinin sil
while (Vb köşesini içeren üçgen varsa)
   Vb köşesini V olacak şekilde düzelt
Vb köşesinin sil
```




Şekil 3.12. Köşe çifti büzme

### 3.1.4. Üçgen büzme algoritması

Kenar büzme işlemine benzer. Üçgen büzme işleminde üçgene ait üç köşe noktası yeni bir noktaya dönüştürülür. Kaldırılan üç kenarla birlikte kaldırılan üç kenara sahip üçgenler de kaldırılır. Yeni noktaya dönüştürülen üç noktaya sahip kaldırılmamış üçgenler yeni noktaya göre tekrar güncellenir (Bkz. Şekil 3.13.) (Luebke, sf:24) (Varshney, sf:12).

Şimdi, üçgen büzme algoritmasının sözde kodlarını inceleyelim (Bkz. Şekil 3.13.).

**Algoritma** :

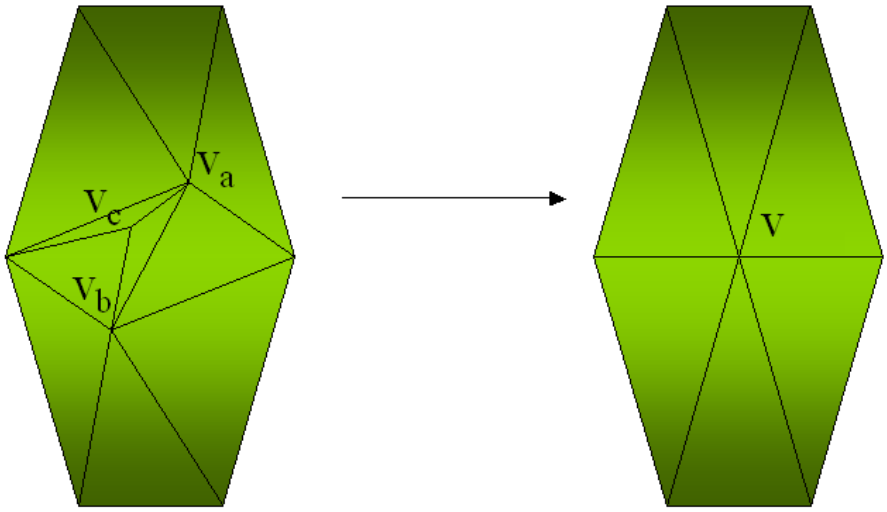
Üçgen büzme algoritması

Girdi : Köşe noktalar kümesi, üçgenler kümesi, kaldırılacak üçgen

Çıktı : Yeni köşe noktalar kümesi, yeni üçgenler kümesi

```
Büzülecek üçgen bulunur

[VaVb] kenarını içeren üçgenleri kaldır

[VbVc] kenarını içeren üçgenleri kaldır

[VcVa] kenarını içeren üçgenleri kaldır

while (Va köşesini içeren üçgen varsa)
```

```
   Va köşesini V olacak şekilde düzelt
Va köşesinin sil
while (Vb köşesini içeren üçgen varsa)
   Vb köşesini V olacak şekilde düzelt
Vb köşesinin sil
while (Vc köşesini içeren üçgen varsa)
   Vc köşesini V olacak şekilde düzelt
Vc köşesinin sil
```




Şekil 3.13. Üçgen büzme

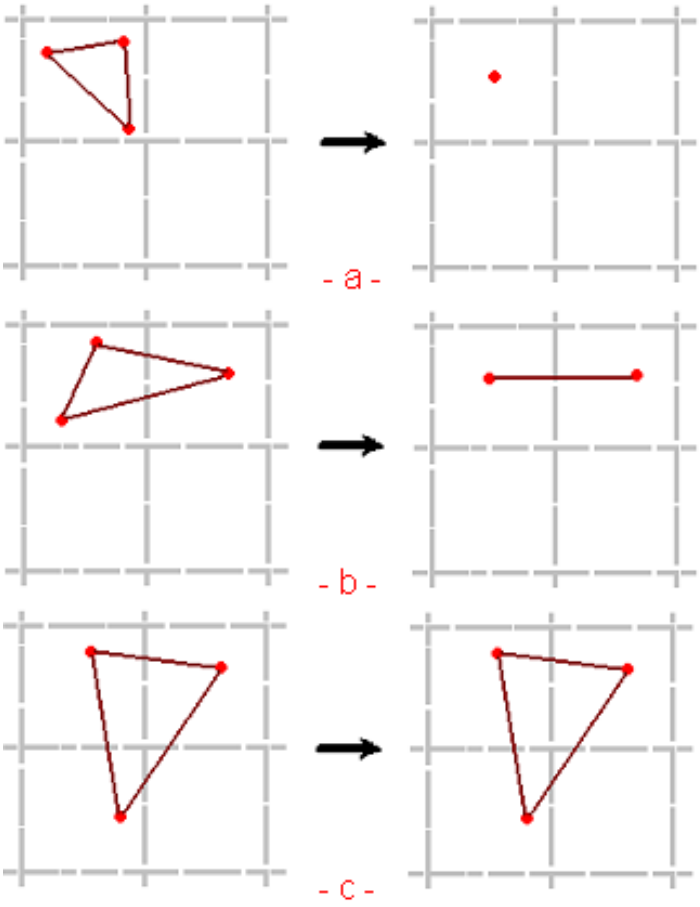
### 3.1.5. Köşelerin birleştirilmesi

Bir grup poligon köşesinin birleştirilmesine dayalı bir algoritmadır. Basitleştirilecek modelin etrafına bir kutu yerleştirilir ve kutu paralel ve dikey doğrularla hücrelere bölünür (Bkz. Şekil 3.16.a). Her bir hücrenin içinde yer alan poligon köşeleri tek bir poligon köşesi olacak şekilde birleştirilir (Bkz. Şekil 3.16.b). Yapılan birleştirme ile modele ait üçgenler tekrar düzenlenir Uygulama için üç farklı seçenek mevcuttur. Seçenekler:

i. Üçgene ait üç köşe aynı hücre içerisinde kalıyorsa bir köşeye dönüştürülür. Üçgen kaldırılır ve hücrede yeni üretilen noktaya dönüştürülür (Bkz. Şekil 3.14.a).

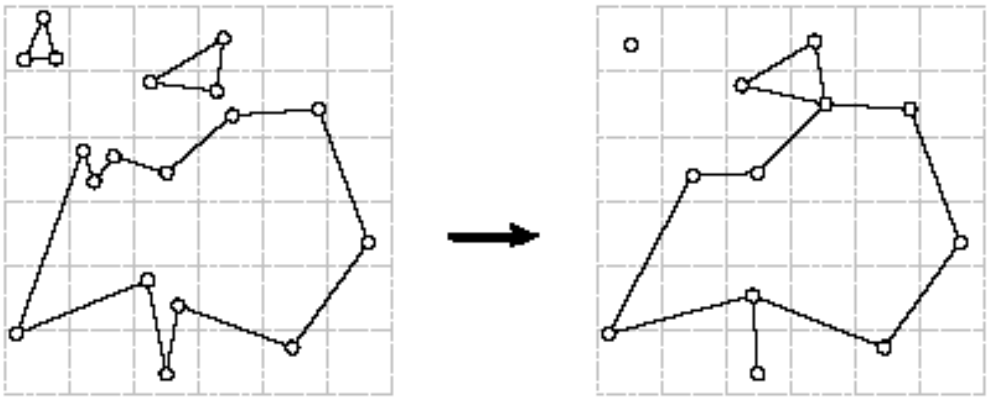
ii. Üçgene ait üç köşenin iki köşesi aynı, bir köşesi ise diğer iki köşe ile farklı hücrede kalıyorsa üçgen doğruya dönüştürülür Üçgen kaldırılır, iki hücredeki yeni köşe noktaları doğru olacak şekilde birleştirilir (Bkz. Şekil 3.14.b).

iii. Üçgene ait üç köşenin üçü de farklı hücrelerde kalıyorsa üçgen özelliği korunmaktadır. Yapılacak işlem sadece üç köşenin içerinde bulundukları üç hücre içerisinde oluşturulan yeni köşe noktalarına göre üçgen köşe noktaları tekrar güncellenir (Bkz. Şekil 3.14.c) (Bkz. Şekil 3.16.c, 3.16.d) (Bkz. Şekil 3.15.) (Güdükbay, sf:3) (Vigrnond sf:10).

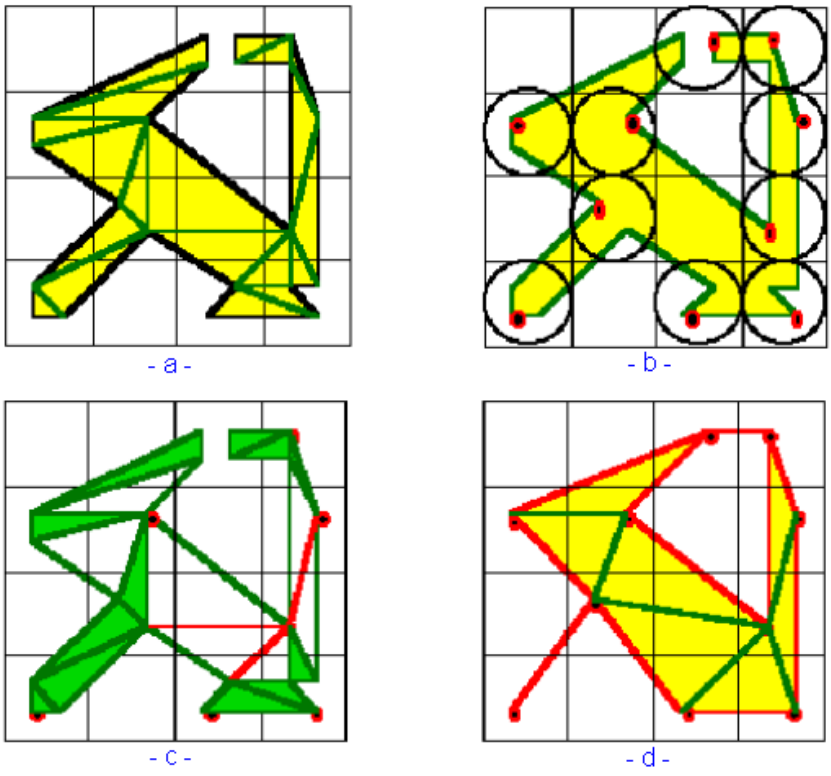



Şekil 3.14. Köşe birleştirme

Bu işlem çok hızlı ve model üzerinde istenilen bütün topolojik değişiklikleri yapabilmektedir. Fakat küçük parçaların büyüklükleri yapılan basitleştirme için geometrik bir hata payı tanımlanabilmesine olanak sağlamaktaysa da elde edilen basitleştirilmiş modelin kalitesi genellikle düşük olmaktadır (Güdükbay, sf:3).



Şekil 3.15. Köşe birleştirme

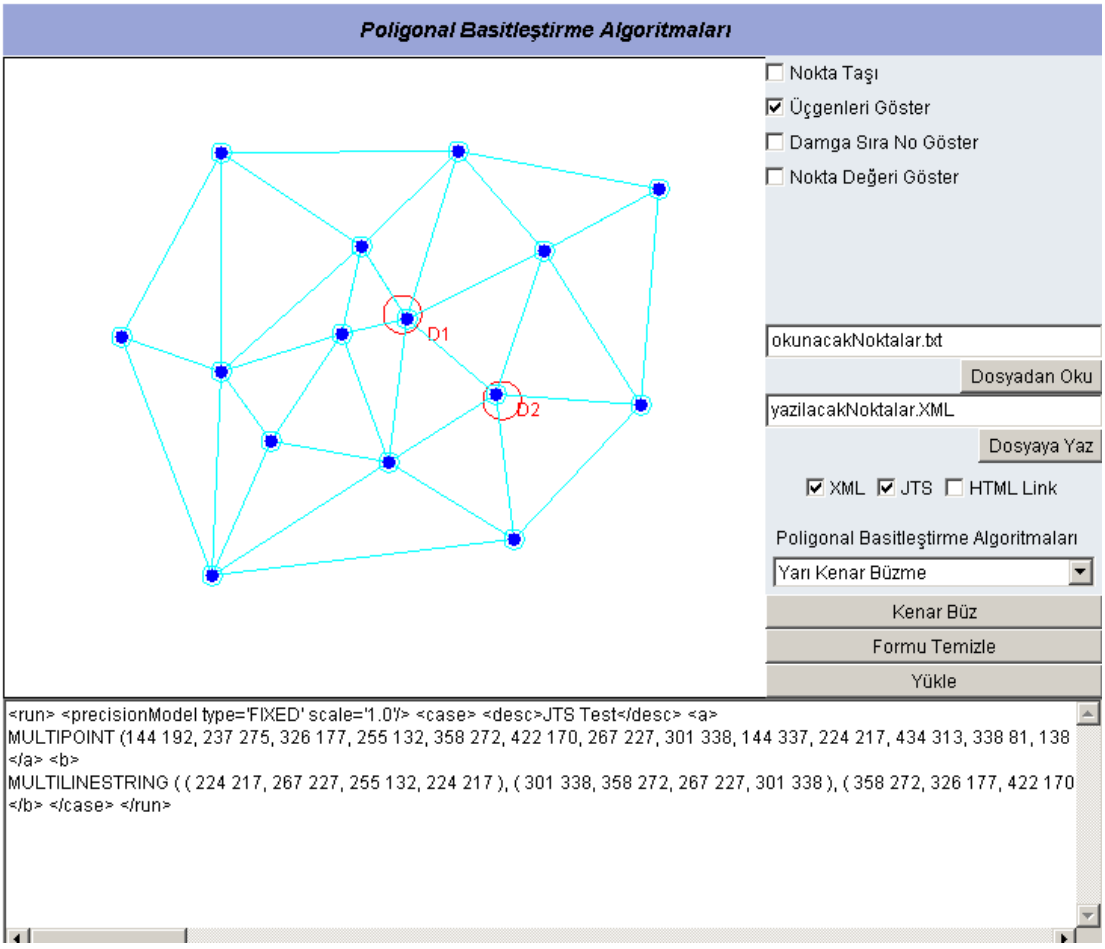



Şekil 3.16. Köşe birleştirme

## 3.2. JTS Test

Üçgen sayısının azaltılmasıyla ilgili algoritmaların daha iyi anlaşılabilmesi için Java applet hazırlanmıştır (Bkz. Şekil 3.17.). Uygulama hazırlanırken JTS Java paketindeki sınıflar ile JTS paketi için hazırlamış olduğumuz ek yama sınıflar kullanılmıştır (Bkz. EK.1). Hazırlanan bu Java uygulamasında girişi yapılan noktaların üçgenleştirilmesi hemen yapılmaktadır. Üçgenleştirme algoritması olarak Delaunay üçgenleştirme algoritması kullanılmaktadır.

Üçgenleştirilmiş bölge üzerinde seçilen köşe ya da köşelerin yardımı ile poligonal basitleştirme algoritmalarının test edilmesi sağlanmıştır (Bkz. Şekil 3.17.).



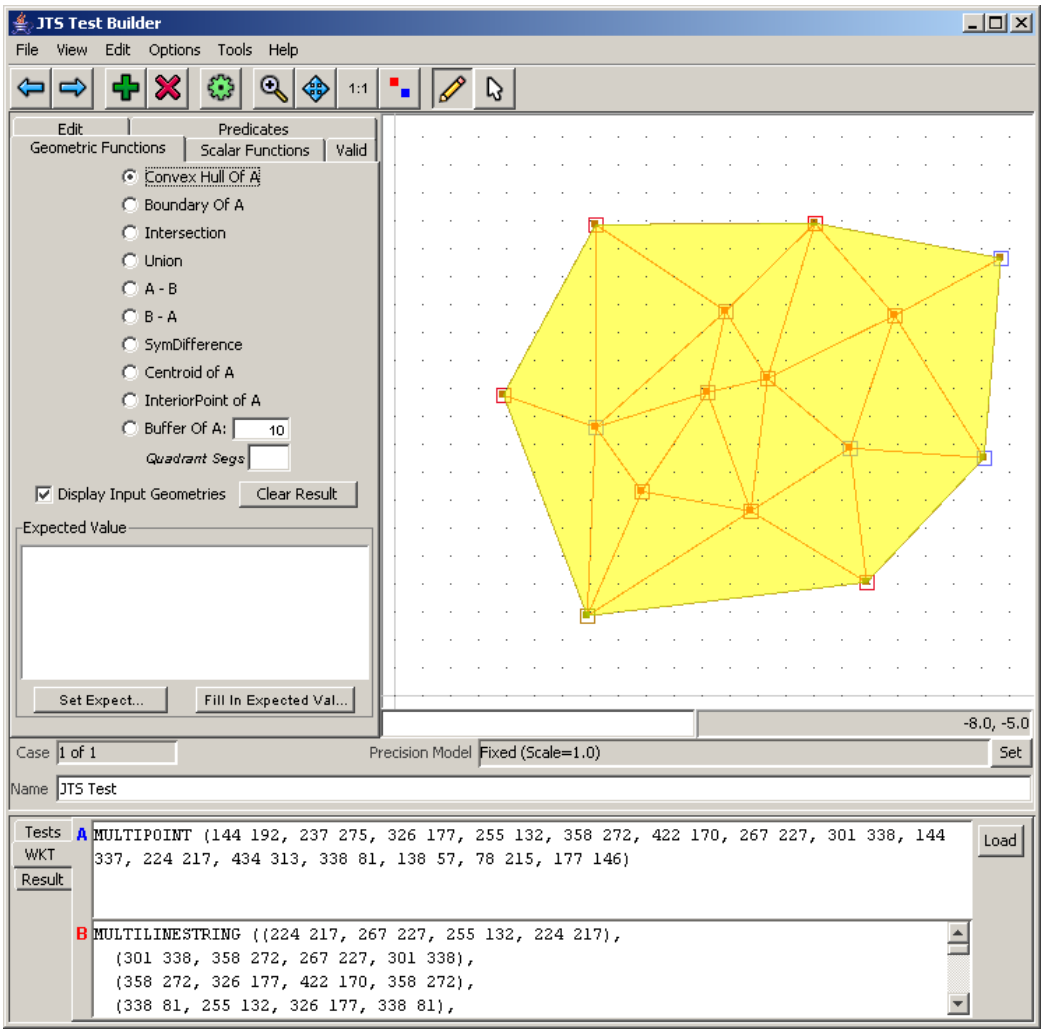

Şekil 3.17. Poligonal basitleştirme algoritmaları ağ uygulaması

Üçgenleştirilmiş bölge, fare imleci yardımı ile nokta ekleme şeklinde elde edilebileceği gibi “Dosyadan Oku” düğmesi ile seçilen dosyanın okunması yöntemiyle de elde edilebilir. Okunan noktalar üçgenleştirilip görüntülenir. Üçgenlerin görüntülenmesi için “Üçgenleri Göster” seçeneğinin seçili olması gerekmektedir.

Üçgenleştirilmiş bölge elde edildikten sonra poligonal basitleştirme algoritmalarından test edilmesi istenen algoritma seçilir. “Köşe Kaldır” algoritması için bir adet köşe seçilir ve “Köşe Kaldır” düğmesine basılır. “Yarı Kenar Büzme”, “Tam Kenar Büzme” ve “Köşe Çifti Büzme” algoritmalarından biri seçilirse iki adet köşe seçilir ve ilgili düğmeye basılır. “Üçgen Kaldır” algoritması seçildiğinde ise aynı şekilde kaldırılması istenen üçgene ait üç adet köşe seçilir ve ilgili düğmeye basılır. Düğmeye basılmasıyla seçili algoritmaya bağlı olarak üçgen sayısı azaltılarak üçgenleştirilmiş bölgenin son hali tekrar görüntülenir.

Uygulamayı hazırlarken JTS Java paketindeki sınıfları kullandığımızı belirtmiştik. JTS paketiyle birlikte JTS Test Builder uygulama programı da gelmektedir. Uygulama programı ile uygulama ekranından fare imleci ile veri girişi yapılabildiği gibi XML dosya türünden veriler okunabilmektedir. Uygulamada A ve B gibi iki farklı nesne verisi değişken olarak hafızada tutulabilmekte, tutulan verilerin üzerinde geometrik fonksiyonların uygulaması yapılabilmektedir. Uygulamadaki verilerin xml ve html dosya türünden kayıt edilmesine de imkan verilmektedir.

Hazırlamış olduğumuz Java applet’inde hafızada tutulan verilerin kayıt edilebilmesi için “Dosyaya Yaz” düğmesi uygulamaya eklenmiştir. “XML” seçeneği seçili ise kaydedilen dosya xml dosya türünden, “XML” seçeneği seçili değil ise txt dosya türünden kayıt işlemi yapılır. Uygulamaya eklenen dosyaya yazma özelliğiyle, verilerin hem txt hem de xml dosya türünden saklanmasına imkan sağlanmıştır. Ayrıca, Java uygulamasındaki verilerin JTS Test Builder uygulamasına aktarılması da sağlanmaktadır. Örneğin, Şekil 3.17’de hafızadaki nesne dizisinin “Dosyaya Yaz” düğmesine basılarak bir xml dosyası oluşturulmuştur. Oluşturulan bu xml dosyası JTS Test Builder’in xml okuma özelliği kullanılarak JTS Test Builder’a aktarılması sağlanmıştır (Bkz. Şekil 3.18.).




Şekil 3.18. JTS Test Builder

# 4. SONUÇLAR

Bu tezde iki boyutlu üçgenleştirme algoritmaları ile üçgen sayısının azaltılması ile ilgili yapılmış çalışmalar derlenip özet şeklinde okuyucuya sunulmuştur. Algoritmaların sözde kodları verilerek algoritmaların çalışma mantığının daha iyi anlaşılması hedeflenmiştir. Bu çerçevede, Java applet'leri ile uygulamaları geliştirilmiş ve sunulmuştur.

Ayrıca, dünya çapında kabul görmüş, coğrafi bilgi sistemleri konusunda kaynak olan Java Topology Suite paketi incelenmiş, üçgenleştirme ve üçgen sayısının azaltılması ile ilgili ek yama kodlar JTS Java paketine eklenmeye müsait bir şekilde kodlanmıştır.

## KAYNAKLAR


Bildirici, İ.Öztuğ ve Selvi, H.Zahit ,2005, Model Genelleştirmesinde Geometri Değişimlerinde Alan Çizgi Dönüşüm Yöntemleri, TMMOB Harita ve Kadastro Mühendisleri Odası 10.Türkiye Harita Bilimsel ve Teknik Kurultayı Mart 2005,Ankara.

Bourke, Paul ,1989, Efficient Triangulation Algorithm Suitable for Terrain Modelling, Pan Pacific Computer Conference, Beijing, China.

Çetin, Nurhan, 2000, Mesh Generation. Computer Based Learning Unit, University of Leeds

De Floriani, Leila ,2005, Mesh simplification. http://www.cs.umd.edu/class/fall2005/cmsc741/simplification-741.pdf. (Erişim tarihi: 16.08.2006)

D'Hondt, Theo, 1997, Computer Graphics. ftp://prog.vub.ac.be/pub/Courses/CGR/PDF.dir/4_PT.pdf (Erişim tarihi: 24.02.2006)

D'Hondt, Theo, 1997, Computer Graphics. ftp://prog.vub.ac.be/pub/Courses/CGR/PDF.dir/5_VD.pdf (Erişim tarihi: 24.02.2006)

Eguchi, Glenn, 2003, Delaunay Triangulations. http://theory.lcs.mit.edu/~indyk/6.838-old/handouts/lec9.pdf. (Erişim tarihi: 27.01.2006)

Garland, Michael ,1999, Quadric-Based Polygonal Surface Simplification, School of Computer Science Carnegie Mellon University.

Garland, Michael, 1999, Multiresolution Modeling: Survey & Future Opportunities, Eurographics'99, State of The Art Reports.

Garey, Johnson, 1996, Triangulation of Monotone Polygon. http://www.cs.ucf.edu/courses/cot5520/lectures.html. (Erişim tarihi: 21.02.2006)

Greiner, Gunther & Girod, B,2004, Principles Of 3d Image Analysis And Synthesis, Kluwer Academic Publishers, Boston-Dodrecht-London

Güdükbay, Ugur ,1998, Çok Çözünürlüklü Modelleme İçin Poligonal Basitleştirme, Sinyal İşleme ve Uygulamaları Kurultayı (SIU'98), Kızılcahamam, Ankara

Hadwiger, Markus ,1999, Geometry over Network Techniques: Mesh Simplification and Multiresolution Data Structures, 4th Research Seminar on Visualization: "Volume Visualization and Interactive Visualization" Winter Semester 1998/1999

Heckbert, Paul S. and Garland, Michael, 1997, Survey of Polygonal Surface Simplification Algorithms, School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213, Tech. Rep. CMU-CS-95-194

Heckbert, Paul & Garland, Michael ,1996, Surface Simplification Using Quadric Error Metrics, Computer Graphics (SIGGRAPH '97 Proceedings)

Hoppe, Hugues ,1996, Progressive Meshes, ACM SIGGRAPH 1996.

Jia, Yan-Bin ,2006, Computational Geometry. http://www.cs.iastate.edu/~cs518/handouts/.(Erişim tarihi: 26.01.2006)

Kreveld, Marc Van and De Berg, Mark and Overmars, Mark and Schwarzkopf, Otfried, 2000, Computational Geometry: Algorithm and Applications

Lambert, Tim, 1998, Experiment with convex hull algorithms. http://www.cse.unsw.edu.au/~lambert/java/3d/index.html.(Erişim tarihi: 16.01.2006)

Luebke, David & Reddy, Martin & Cohen, Jonathan & Varshney, Amitabh & Watson, Benjamin and Huebner,2002, Level of Detail For 3D Graphics3D, Morgan Kaufmann

Melax, Stan, 1998, A Simple, Fast and Effective Polygon Reduction Algoritm, Game Developer Magazine, University of Alberta

Miu, Allen ,2003, Voronoi Diagrams. http://theory.lcs.mit.edu/~indyk/6.838-old/handouts/lec8.pdf. (Erişim tarihi: 27.01.2006)

Mount, Dave ,2005, Delaunay Triangulations: Incremental Construction. http://www.cs.umd.edu/class/fall2005/cmsc754/Lects/lect14.pdf. (Erişim tarihi: 14.02.2006)

Mukherjee, Amar ,2003, Computational Geometry, Triangulation of Monotone Polygon. http://www.cs.ucf.edu/courses/cot5520/lectures.html. (Erişim tarihi: 21.02.2006)

O'Rourke, Joseph, 1998, Computational Geometry In C Second Edition, Cambridge University Press, Cambridge.

Ostrovsky, Yaron & Berman, 2005, Computational Geometry. http://www.cs.huji.ac.il/~compgeom/slides/CG-tirgul07-2spp.pdf.(Erişim tarihi: 21.02.2006)

Ottmann, Thomas, 2003, Computational Geometry. http://download.informatik.uni-freiburg.de/lectures/GeometrischeAlgorithmen/2005-2006WS/Slides/lecture_poly_triangulation_changed.pdf. (Erişim tarihi: 21.03.2006)

Quadros, William ,2003, Scientific Computing. http://www.cs.cmu.edu/afs/cs/user/glmiller/public/computational-geometry/15-859J-F03/Notes/Sept-22/notes12.pdf. (Erişim tarihi: 21.02.2006)

Puppo, E. and De Floriani L. ,2004, Mesh simplification (part three): simplification algorithms. http://www.aimatshape.net/resources/v-lectures/meshsimplification/DISI-simplification-part3.pdf. (Erişim tarihi: 16.08.2006)

Rasit Bin Muhammed. ,1993, Computational Geometry Lecture Notes. http://www.personal.kent.edu/~rmuhamma/Compgeometry/compgeom.html.(Erişim tarihi: 14.02.2006)

Shewchuk, Jonathan ,1999, Meshing and Triangulation in Graphics, Engineering, and Modeling. http://www.cs.berkeley.edu/~jrs/mesh/1/lecture.html.(Erişim tarihi: 18.01.2006)

Skiena, Steven S. & Revilla, Miguel, 2003, Programming Challenges, Springer Verlag

Smid, Michiel ,2003, Computing the convex hull of a planar point set. http://www.scs.carleton.ca/~michiel/lecturenotes/ALGGEOM/convexhull.pdf. (Erişim tarihi: 01.09.2006)

Stewart, Colin ,2003, Polygon Triangulation. http://www.sfu.ca/~cjs/triangulation. (Erişim tarihi: 14.02.2006)

Sunday, Dan, 2004, Convex Hull of a 2D Simple Polyline. http://softsurfer.com/Archive/algorithm_0203/algorithm_0203.htm. (Erişim tarihi: 06.06.2006)

Suri, Subhash, 2002, Polygon Triangulation. http://www.cs.ucsb.edu/~suri/cs235/Triangulation.pdf. (Erişim tarihi: 27.01.2006)

Turk, Greg & Lindstrom, Peter,2003, Image Driven Simplification. http://www.cs.ualberta.ca/~anup/Courses/604/Presentation_files/P3.pdf. (Erişim tarihi: 24.07.2006)

Varshney, Amitabh ,2003, Level Of Detail Management for 3D Games. http://lodbook.com/course/2003/Varshney_GeneratingLOD.ppt. (Erişim tarihi: 04.08.2006)

Vigrnond, Edward Joseph ,1997, Electrical Coupling Mechanisms of Excitable Cells. http://www.collectionscanada.ca/obj/s4/f2/dsk2/ftp03/NQ28309.pdf. (Erişim tarihi: 24.07.2006)

Vlasic, Daniel ,2003, Polygon Triangulation. http://theory.lcs.mit.edu/~indyk/6.838-old/handouts/lec4.pdf. (Erişim tarihi: 27.01.2006)

Yanalak,M., 1997, Sayısal Arazi Modellerinden Hacim Hesaplarında En Uygun Enterpolasyon Yönteminin Araştırılması ,Doktora Tezi ,İTÜ Fen Bilimleri Enstitüsü, İstanbul.

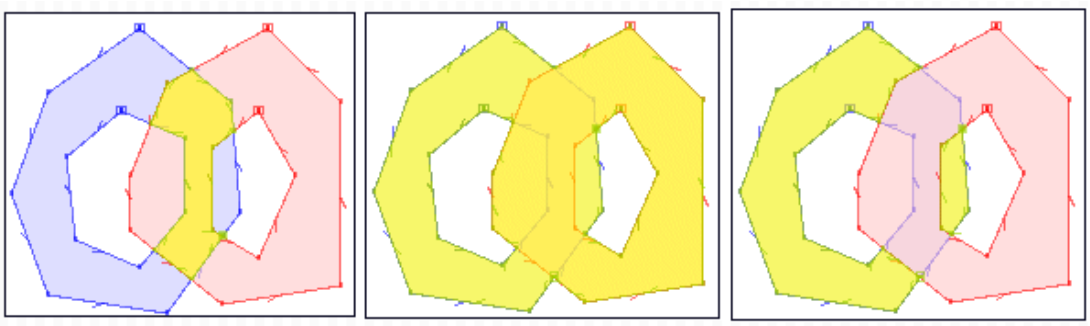
## EK 1.JAVA TOPOLOGY SUITE

Java Topology Suite (JTS) coğrafi bilgi sistemleri alanında kabul görmüş bir Java paketidir. JTS paketi, Open GIS Konsorsiyum tarafından 1999 tarihinde belirlenmiş olan standartlarla tamamen uyumlu bir pakettir. JTS Java paketi, iki boyutlu düzlemde doğrusal geometrik modellemelerin tanımlanmasında, konumsal veri işlemlerinde ve algoritmalar üzerinde tam ve sağlam bir destek sunmaktadır. Ayrıca, konumsal veritabanları için sorgulama imkanları da sunmaktadır.

JTS Java paketi ilk kez 30 Mayıs 2001 tarihinde ortaya çıkmış olup, 2004 yılına kadar altı kez sürüm güncellemesi yapılmıştır. 22 Eylül 2004 tarihinde JTS sürüm 1.5 kullanıcıya sunulmuştur. En son sürüm değişikliği ise 22 Haziran 2006 tarihinde yapılmış ve 1.7.2 sürümü kullanıma sunulmuştur. Web adresi olarak http://www.vividsolutions.com/jts/JTSHome.html adresi kullanılmaktadır. Vivid Solutions, Inc. firmasının sorumluluğunda çalışmalarına devam edilen JTS Java paketinin geliştirme grubunda Martin Davis, Jonathan Aquino, David Skea gibi isimler mevcuttur.

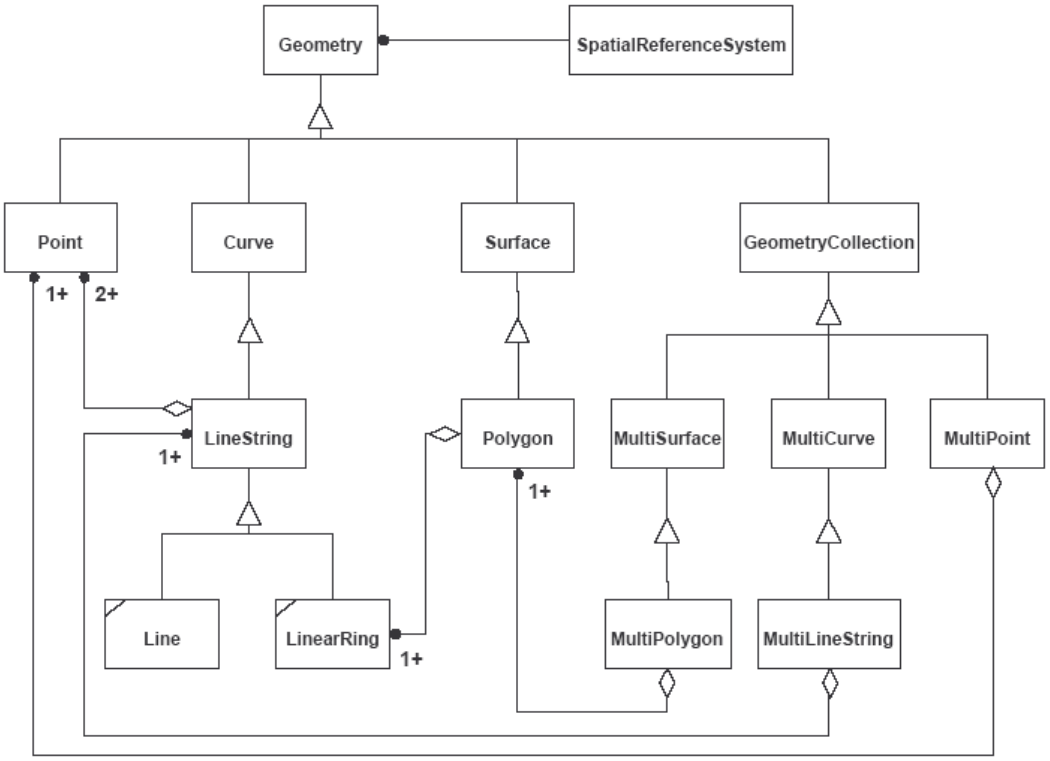
JTS Java paketi veri giriş ve veri çıkışlarında bilinen bütün giriş/çıkış text türlerini desteklemektedir. Ayrıca, bilgisayar grafiği işlemlerinde çok sağlam ve sıhhatli sonuçlar vermektedir. Modellemeler üzerindeki kontrollerde rahat bir kullanım kolaylığı sunmaktadır. İki boyutlu düzlemde konumsal veri işlemlerinde (üzerinde, kesişiyor, .. gibi) ve konumsal analiz metotlarında (kesişim, birleşim, .. gibi) tam, eksiksiz ve birbiri ile tutarlı işlemler gerçekleştirmektedir (Bkz. Şekil 1.).

JTS paketinde, her türden geometrik nesnenin tanımlanabilmesine imkan veren sınıflar tanımlanmış ve kullanıma sunulmuştur. Ayrıca, bütün veri türlerini kapsayacak şekilde Geometry sınıfı adında bir üst sınıf tanımlanmıştır. JTS paketindeki hiyerarşide Geometry sınıfı en üst seviyeyi tutmaktadır (Bkz. Şekil 2.). Bütün nesneler Geometry sınıfı ile de ifade edilebilmekte ve kullanılabilmektedir. Geometry nesnesi ile tanımlanan veri kümeleri üzerinde birleşim, kesişim, fark,.. gibi küme işlemleri uygulanabilmektedir.



Kesişim Birleşim Fark

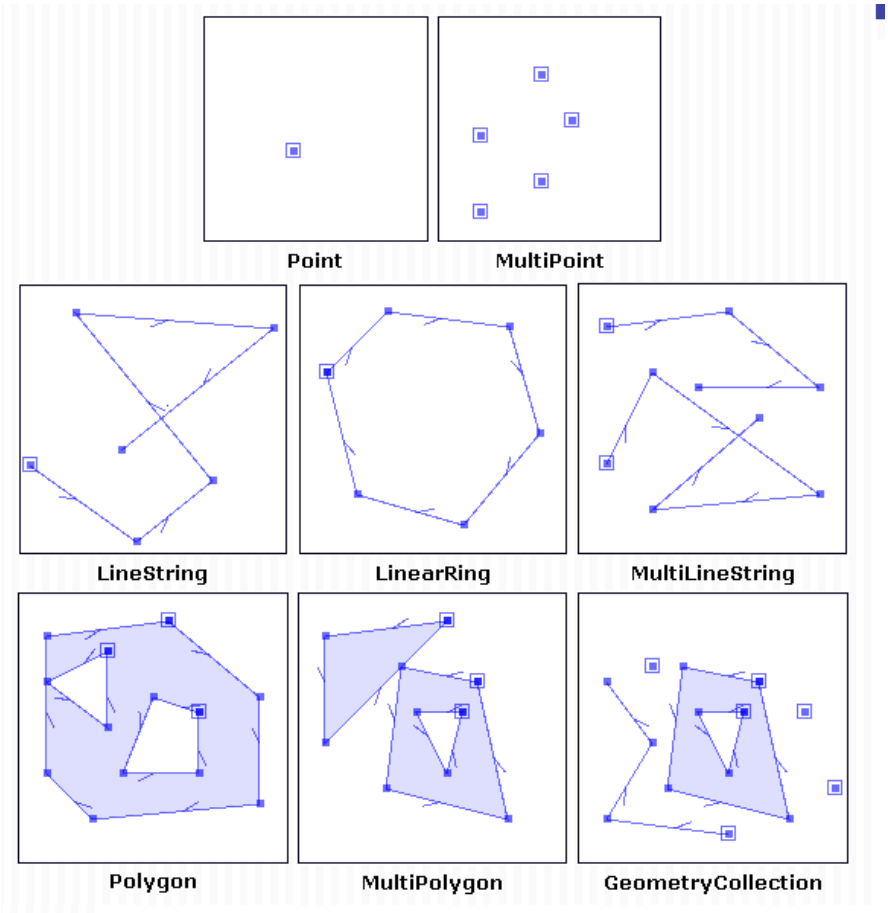
Şekil 1. Veri işlemleri




Şekil 2. Geometrik sınıf hiyerarşisi

Bilgisayar grafiğinde ve grafik uygulamalarında nesneleri ifade etmek için kullanılabilecek en küçük nesne noktadır. JTS Java paketinde (x, y) noktasını ifade etmek için Coordinate sınıfı tanımlanmıştır. Paket içerisinde Coordinate sınıfı kullanılarak yeni sınıflar üretilmiştir. Tanımlanan sınıflardan bazıları Point,

MultiPoint, Line, LineString, Polygon, MultiPolygon, Geometry, .. sınıflarıdır (Bkz. Şekil 3.).

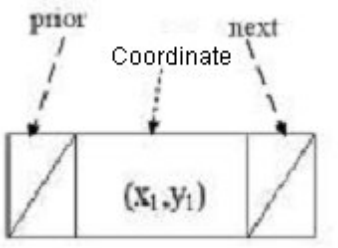



Şekil 3. Geometrik sınıflar

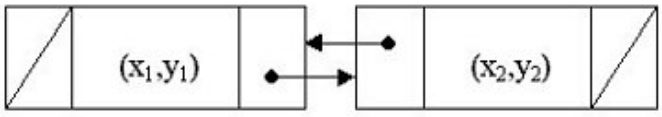
Tezimizde JTS Java paketinin 1.7.2 sürümü kullanılmıştır. Uygulama geliştirilirken JTS Java paketi içinde eksik olan üçgenleştirme ile ilgili yardımcı sınıflar tanımlanmıştır. Tanımlanan sınıflar coordinateNode, coordinateNodeList, triTriangle, triTriangleNode ve triTriangleNodeList sınıflarıdır.

JTS Java paketiyle nokta verisinin tanımlanması için Coordinate sınıfı kullanılmaktadır. Birden fazla aynı türden veriyi bilgisayar hafızasında tutmak için ya dizi kullanılmalı ya da gösterge zincirleri kullanılmalıdır. Gösterge zincirleri poligonal basitleştirme işlemlerinde daha kullanışlı olacağından Coordinate zinciri için coordinateNodeList ve üçgenler zinciri için ise triTriangleNodeList sınıfları hazırlanmıştır.

JTS paketi için tanımlanan coordinateNode sınıfı, noktalar dizisi oluşturabilmek için tanımlanmış bir sınıftır. Girişi yapılan her nokta için önce coordinateNode nesnesi tanımlanıp, önceki ve sonraki göstergeler ile birbirlerine bağlayarak uç uca bir liste oluşturulur (Bkz. Şekil 4.) (Bkz. Şekil 5.). Bu türden gösterge zincirlerinin özelliklerinden biri eleman sayısının önceden belirlenmemesi ve dolayısıyla eklenecek eleman sayısının önceden sınırlandırılmasının önüne geçilmiş olmasıdır. coordinateNode sınıfının tanımlaması şu şekildedir;




Şekil 4. Gösterge yapısı




Şekil 5. Gösterge yapısı

```
public class coordinateNode extends Object
public coordinateNode prior;  // önceki nesne
public coordinateNode next;   // sonraki nesne
public Coordinate coordinate; // Coordinate sıfını değeri
  // Argümansız yapılandırıcı
public coordinateNode()
  // Coordinate nokta sınıfı değer alan yapılandırıcı.
public coordinateNode(Coordinate P)
}
```

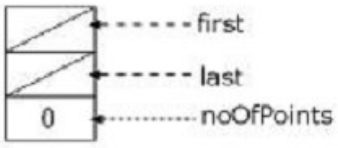
coordinateNodeList sınıfı, noktalar kümesi oluşturmak için kullanılır. coordinateNodeList nesnesi noktalar kümesinin ilk ve son elemanı ile eleman sayısını tutabilen bir nesnedir (Bkz. Şekil 6.). coordinateNodeList nesnesinin tanımlaması ise şu şekildedir;

```
public class CoordinateNodeList extends Object
public coordinateNode first; //Listenin ilk değerini tutar.
public coordinateNode  last; //Listenin son değerini tutar.
public int       noOfPoints; //Listedeki eleman sayısı
  //Argümansız yapılandırıcı
public coordinateNodeList().
  //Argümanlı yapılandırıcı.
public coordinateNodeList(Coordinate[] coorL, int countT)
}
```

Coordinate nokta nesnelerini bir küme oluşturacak şekilde kullanmak istersek şu adımları izlemek yeterli olacaktır (Bkz. Şekil 6.3.). Öncelikle, bellekte ilk ve son işaretçileri boş ve eleman sayısı sıfır olan bir coordinateNodeList nesnesi oluşturulmalıdır. Bu işlem için;

```
coordinateNodeList coordinates = new coordinateNodeList ();
```

kodlanır.

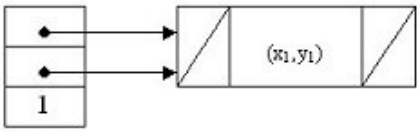



Şekil 6. Pointer yapısı

Verilen $(x_1,y_1)$ noktasını listeye eklemek için;

```
coordinates.addLast(new Coordinate(x1,y1));
```

kodlanır (BKz. Şekil 7.).

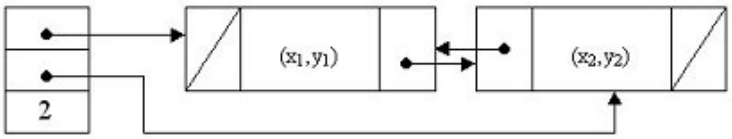



Şekil 7. Listeye ilk noktayı ekleme

Verilen $(x_2,y_2)$ noktasını listeye eklemek için (Bkz. Şekil 8.);

```
coordinates.addLast(new Coordinate(x2,y2));
```

kodlanır.

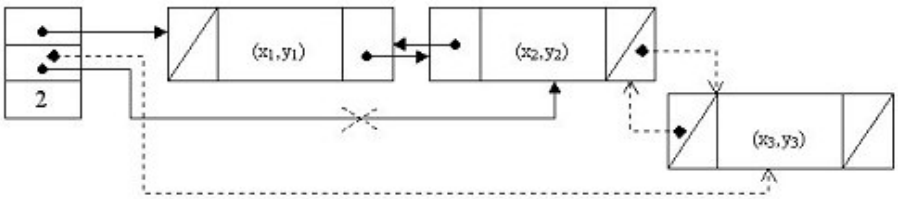



Şekil 8. Listeye ikinci noktayı ekleme

Verilen $(x_3,y_3)$ noktasını listeye ekleme işlemini biraz daha görselleştirecek olursak. İşlem için;

```
coordinates.addLast(new Coordinate(x3,y3));
```

kodlanır. Ekleme işlemi bittiğinde eleman sayısı 2 den 3 e dönüştürülecektir (Bkz. Şekli 9).




Şekil 9. Listeye üçüncü noktayı ekleme

Java appletleri için kullanılacak nokta dizileri için kullanılacak gerekli fonksiyonlarda bu sınıf içerisinde tanımlanmıştır.

Pakete eklenen sınıflardan `coordinateNodeList` sınıfı nokta değerlerinin gösterge listeleri olup, üçgenleştirme için gerekli yardımcı fonksiyonlar tanımlanmıştır. Sınıf içerisinde tanımlanmış fonksiyonlardan bazılarını şunlardır;

```
//Gelen Coordinate P nesnesini liste sonuna ekler.
public void addLast(Coordinate P)

//Gelen Coordinate P nesnesini tekrar etmemek koşulu ile
//liste sonuna ekler.
public void addLastUnique(Coordinate P)

//Listeyi boşaltır.
public void delAll()

//Listedeki Coordinate P verisini siler.
public void delOneCoordinate(Coordinate P)

//Gelen Coordinate P verisini istenen damgadaki yere atar.
public void updateIndexValue(int index, Coordinate P)

//Bir Coordinate P nesnesinin listedeki damgasının değerini
//geri döndürür.
public int getCoordinatePos(Coordinate P)

//Bir Coordinate P nesnesinin listedeki damgasının değerini
//epsilon hata payı ile bulup geri döndürür.
public int getNearestPtsPos(Coordinate P, int epsilon)

//Listeden istenen damgada yer alan Coordinate nesnesini geri
//döndürür
public Coordinate getIndexValue(int index)

//Listenin kopyasını çıkartır
public coordinateNodeList copyPolygon()

//Göstergedeki noktaları çizer.
public void drawPolygon(Graphics gg, boolean showPtsTitle
          , boolean showPtsValue, Color pointColor
          , Color lineColor, Color textColor
          , boolean joinFirstLast, boolean drawLines
          , boolean drawPoints, boolean fillPoints
          , int xSize, int ySize, int xGolge, int yGolge )

//Listedeki noktaların arasındaki bölgeyi boyar.
public void fillPolygon(Graphics gg, Color fillColor )

//Noktanın konveks çok kenarlının içinde olup olmadığını
//kontrol eder.
public boolean pointInConvexPolygon(Coordinate P)
```

```
//Göstergedeki noktaların istenen damgaya kadar ki kısmını
//Coordinate dizisini olarak geri döndürür
public Coordinate[] toCoordinateArray(int aktifPtsId)

//Göstergedeki noktaların Delaunay üçgenleştirmesi ile elde
//edilen üçgenler dizini geri döndürür
public triTriangleNodeList getDelaunayTriangles()

//Konveks kabuğu geri döndürür
public coordinateNodeList getConvexHull()

//Poligon'nun basit poligon olup olmadığı kontrol edilir
public boolean isSimplePolygon(Coordinate newPts
          , Coordinate chkPts, int chkLineIndx, int chkFrk)
```

JTS paketine eklenen `coordinateNodeList` sınıfını inceledik. JTS Java paketine eklenen diğer bir sınıf ise `triTriangleNodeList` sınıfıdır. Bu sınıfta üçgenler listesi tutulması, tutulan üçgenler üzerinde üçgenleştirme ve üçgen sayısının azaltılması ile ilgili fonksiyonlar tanımlanmış olup, sınıf içerisinde tanımlanmış fonksiyonlardan bazılarını şunlardır;

```
//Gelen T nesnesini liste sonuna ekler.
public void addLast(triTriangle T)

//Listeyi boşaltır.
public void delAll()

//Listedeki son elemanı siler.
public void delLast()

//Listedeki belli bir damgadaki elemanı siler.
public void delOneTriangle(int index)

//T değerini istenen damgadaki yere atar.
public void updateTriangleIndexValue(int index,triTriangle T)

//Listeden istenen damgadaki nesnesini geri döndürür.
public triTriangle getIndexValue(int index)

//Listenin kopyasını çıkartır.
public triTriangleNodeList copyTriangleList()

//Yeni liste aktif liste üzerine eklenir
public void addOverList(triTriangleNodeList eklenenList)

//P köşesini içeren üçgenler kümesini döndürür
public triTriangleNodeList getTrianglesForOnePts
          (Coordinate P)

//Listeyi çok kenarlı olarak çizer.
```

```
public void drawTriangles(Graphics gg, boolean drawTriangle
          , boolean drawCircle, boolean drawLines
          , Color pointColor, Color lineColor
          , Color textColor, Color originColor
          , boolean drawPoints, boolean fillPoints
          , int xSize, int ySize, int sizeO
          , int golgeX, int golgeY)

//P1 ve P2 köşelerini kapsayan üçgen damgası geri döner.
//hariciIndex = 0 ise ilk bulunan damga, hariciIndex > 0 ise
//hariciIndex ile farklı damga dondurulu
public int getIndexValueForTwoPts(int hariciIndex
          , Coordinate P1, Coordinate P2)

//Yeni eklenen nokta üçgenleştirilmiş bölgenin dışında ise
//kullanılacak Delaunay üçgenleştirmesi algoritması
public void addOneCoordinateOutDelaunayTriangle(
            Coordinate nwPts, coordinateNodeList liste
          , int aktifPtsId, coordinateNodeList flipClm01
          , coordinateNodeList flipClm02)

//Yeni eklenen nokta üçgenleştirilmiş bölgenin içinde ise
//kullanılacak Delaunay üçgenleştirmesi algoritması
public void addOneCoordinateInDelaunayTriangle(
            Coordinate nwPts, coordinateNodeList liste
          , int aktifPtsId)

//Delaunay üçgenleştirmesi için kenar dönüşümü
public void flipEdgeFor2Pts(Coordinate nwPts
          , Coordinate edgeA, Coordinate edgeB)

//Gelen noktanın üçgenlerin içindemi dışında mı olduğu
//kontrol edilir. Nokta üçgen içerisinde ise damgası,
//dışarıda ise 0 döner
public int getCoordinatePosition(Coordinate nwPts)

//Üçgenlerin Coordinate kümesini döndürürü
public coordinateNodeList getCoordinateList()

//Yarı kenar büzme
public void meshYariKoseBuzme(Coordinate ptrP1
          , Coordinate ptrP2, coordinateNodeList coorList)

//Tam kenar büzme
public void meshTamKoseBuzme(Coordinate ptrP1
          , Coordinate ptrP2, coordinateNodeList coorList)

//Üçgen büzme
public void meshUcgenBuzme(triTriangle triGel
          , coordinateNodeList coorList )
```

Hazırlamış olan bu Java sınıfları ile üçgenleştirme ve üçgen sayısının azaltılması ile ilgili uygulama geliştirmek çok daha kolay olmaktadır. Uygulama hazırlanırken üçgenleştirilecek bölgedeki nokta değerlerinin girilmesi, girilen değerlerin gösterge listesine eklenmesi ve ilgili fonksiyonların çağrılması yeterli olacaktır.