**KOCAELİ ÜNİVERSİTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**Programlama Laboratuvarı I- I. Proje**

**Minumum Çevreleyen Çember**

**(Minimum Enclosing Circle (MEC))-B Spline**

*Gökçe Yılmaz yilmaz.gokce.tr@gmail.com*

**Özet**

Bu projedeki amaç kullanıcı tarafından girilen noktalar, dosyadan okunarak tamsayı koordinatlı 2 boyutlu bir düzlemde verildiğinde tüm noktaları içeren minimum yarıçaplı çevreleyen çember çizdirilmesidir. Oluşturulan çemberin yarıçapı ve merkezi çıktı olarak verilmekle birlikte çemberin üzerinde gösterildi. Verilen noktaların en yakınından geçen Spline eğrisi çizdirildi.

**I. Giriş**

Projede C programlama dili ve Codeblocks geliştirme ortamı kullanıldı. Arayüz tasarımı için Allegro kütüphanesinden yararlanıldı. Allegro kütüphanesinin gerekli uzantıları çalışmanın bulunduğu klasöre eklendi. Allegro’da yazı tipi kullanımı için *arial.ttf* uzantısı klasöre kopyalandı.Aynı klasörün içine kullanıcının veri girişi yapacağı metin dosyası eklendi ve veriler bu dosyadan okundu.

**II. Yöntem**

İlk olarak allegro yardımıyla 1000\*1000 büyüklüğünde *Minimum Enclosing Circle-1.Proje* adlı bir pencere açtık. Allegronun *al\_draw\_line* eklentisini kullanarak koordinat sistemini yirmişer aralıklarla böldük, *al\_draw\_text* eklentisi ve itoa fonksiyonu yardımıyla *x-y* doğrularını ölçeğe göre numaralandırdık. Kullanıcının değer girdiği *girilennoktalar.txt* metin dosyasını açtık ve kullanılacak olan noktaların değerlerini okuyarak noktalar adlı matrise atadık. Kullanıcının istediği sayıda değer girebilmesi için matrisin ilk değer indisinin boyutunu 1000 olarak tanımladık.

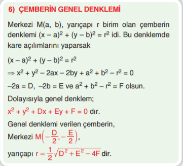
Minimum çevreleyen çember probleminin çözülmesinde Welzl algoritması ve üç noktası verilen çemberin denkleminden yararlandık. Emo Welzl tarafından 1991 yılında en küçük çevreleyen çember için tanımlanan recursive bir algoritmadır. Welzl algoritması basit tanımlamayla rastgele seçilen noktadan oluşturulan minimum çemberin tüm noktaları kapsayana kadar büyütülmesine dayanır.

Öncelikle *x*, *y* ve *r* değerlerinin kullanımının kolay olması için *circle* adında bir struct tanımladık ve daha sonra fonksiyonlarda bunu kullandık. İki nokta arasındaki uzaklık formülünü *mesafe\_bul* fonksiyonunda tanımladık ve tekrar tekrar formülü yazmak yerine bulmak istediğimiz değerleri çalışma süresince bu fonksiyona yolladık.

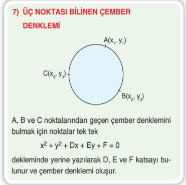
Main fonksiyonda tuttuğumuz noktaları ilk olarak *cember\_bul* fonksiyonuna yolladık. Bu fonksiyonda girilen ilk iki değeri varsayılan çemberin merkezinin belirlenmesinde kullandık, *mesafe\_bul* fonksiyonuna iki noktayı göndererek bulduğumuz çap değerini ikiye bölerek yarıçapı elde ettik. Varsayılan çemberin diğer noktaları kapsayıp kapsamadığını test etmek için *icindemi* adlı bir değişken tanımladık ve bu değişkende üçgendeki *a^2* + *b^2 = c^2* formülü kullandık. Eğer *icindemi* değişkeni sıfırdan büyükse belirtilen noktaları varsayılan çemberin dışında olduğundan *teknokta\_disarda* olarak adlandırdığımız fonksiyona gönderdik.

Nokta gönderme işleminden sonra, *teknokta\_disarda* fonksiyonunda dışarda kalan nokta ve okunan ilk nokta arasındaki mesafe çap olarak varsayılarak hesaplandı. Bu işlemden sonra tekrar *icindemi* değişkeniyle test edildi ve hala dışarda kalan bir nokta varsa *ikinokta\_disarda*

fonksiyonuna gönderildi. Dışarda kalan birinci ve ikinci nokta arası çap olarak varsayıldı. Burada da tekrar aynı işlemle yapıldı ve kapsama test edildi. Hala dışarda kalan nokta varsa çemberin üzerinde üç nokta var demek olduğu için *ucnoktasi\_verilencember* fonksiyonuna birinci, ikinci ve üçüncü noktalar gönderildi. Bu fonksiyonda üç noktası verilen çemberin genel denkleminden yararlanıldı. Yararlanılan denklemler Resim 1 ve Resim 2’de belirtilmiştir. Gönderilen noktalar hangi fonksiyonda doğru sonuca ulaştıysa o fonksiyondaki *x*, *y* ve *r* değerleri döndürüldü ve main fonksiyona gönderildi. Main fonksiyonda çember, çemberin merkezi ve yarıçap çizdirildi. Çemberin yarıçapı çizdirilirken başlangıç *x*, *y* değerleri çemberin merkezi olarak alındı. Bitiş *x*, *y* değerlerinde ise y değeri, merkez noktasının y değeri; x değeri ise merkez noktasının x değerine yarıçap uzunluğu eklenmesiyle oluşan x değeri olarak *(ce.r+ce.xeks)* alındı.

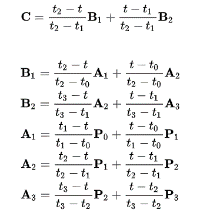


Resim : Çemberin Genel Denklemi

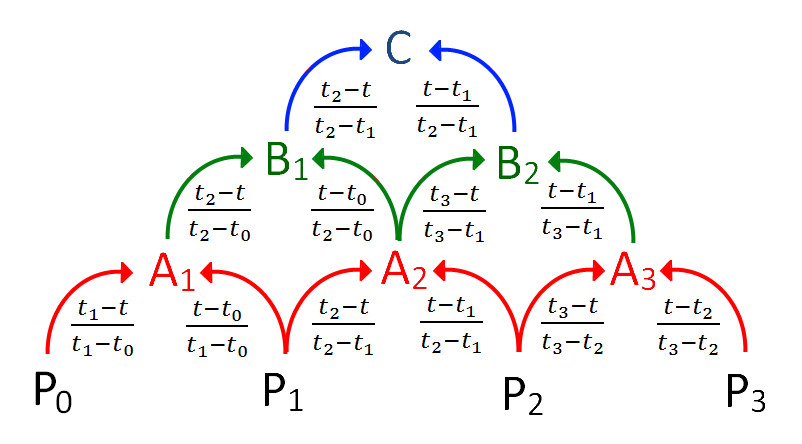


Resim : Üç Noktası Verilen Çember Denklemi

Spline eğrisini çizdirirken B-Spline eğrisini çizdirmekte ve matematiksel kısmı koda dökmekte zorlandığımız için Catmull-Rom Spline eğrisini kullandık. Catmull-Rom Spline eğrisi tanımlana dört kontrol noktası üzerinden geçen bir eğridir. *Pi= [xi yi]^T*  bir noktayı gösterir ve *P0*, P1, P2, P3 kontrol noktalarından geçen bir C eğrisinin matematiksel denklemi Şekil 1 ve Şekil 2’deki gibi açıklanabilir.



Şekil 1

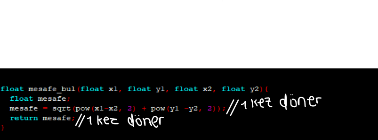


Şekil 2

Zaman karmaşıklığı analizi kısmına gelmeden önce kısa bir tanımlama yapmak gerekirse programın ya da fonksiyonun işlevini tam anlamıyla yerine getirebilmesi için her işlemden kaç kere yapması gerektiğini gösteren bir bağıntıdır. Big O complexity analysis ise fonksiyonların input size’ı büyüdükçe oluşacak karmaşıklığı göstermek için kullanılır. Fonksiyonun input size’ı büyüdükçe küçük kalan big o notasyonu makbüldür. Eğer input size büyürken big o da çok fazla büyüyorsa var ise alternatiflerini kullanmamız gerekir [[1]](http://cagataykiziltan.net/programin-calisma-hizi-ve-algoritma-verimliligi/zaman-karmasikligi-ve-buyuk-o-notasyonu-time-complexity-and-big-o-notation/).

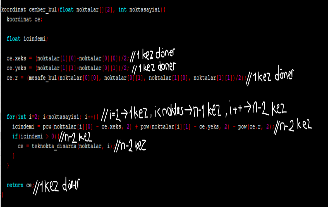
 Big o bulunurken time complexity bağıntısındaki en büyük artışı yöneten terim alınır diğerleri göz ardı edilir çünkü input size sonsuza giderken bu büyümeyi yönetecek olan en büyük değişkenli derimdir.

Çemberi çizdirirken kullandığımız algoritmanın zaman karmaşıklığını adım adım hesapladık. Öncelikle *mesafe\_bul* fonksiyonunun karmaşıklığını Kod Parçası 1’de görüldüğü gibi hesapladık. Sadece bir kez dönen değer atama ve return kısmı olduğu için zaman karmaşıklığı *T(n)=2* olarak alınır.



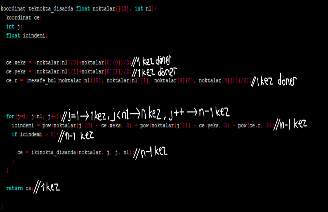
Kod Parçası 1

İkinci olarak *cember\_bul* fonksiyonunu hesapladık. Atama değerleri birer kez döndü. For döngüsü içerisinde *i=2*’den başladığı *i<noktasayisi*, *n-1* kez ve *i*++, *n-2* kez döner. For döngüsü içindeki atamalar ve if yapısı ise *i*++ ile aynı olacak şekilde *n-2* kez döner. Toplamda zaman karmaşıklığı *T(n)= 3+1+(n -1) + (n-2) + 3* \**(n-2 ) + 1* işlemleri yapılarak *T(n)=5n-4* olarak hesaplanır.



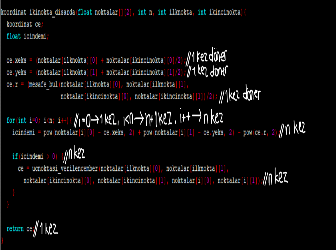
Kod Parçası 2

Üçüncü olarak *teknokta\_disarda* fonksiyonunu hesapladık. Değer atamalar birer kez döndü. For döngüsü içinde *j=1* olduğu için *j=1*, 1 kez; *j<n1*, *n* kez ve *j*++, *n*-1 kez döndü. Döngünün içindeki değer atamalar ve if yapısı da *j ++* ile aynı olacak şekilde *n-1* kez döndü. Toplamda zaman karmaşıklığı *T(n)=3+1+n+(n-1)+3\*(n-1)+1* işlemleri yapılarak *T(n)=5n+1* olarak hesaplanır.



Kod Parçası 3

Dördüncü olarak *ikinokta\_disarda* fonksiyonunu hesapladık. Değer atamalar birer kez döndü. For döngüsü içinde *i=0* olduğu için *i=0*, 1 kez; *i<n*, *n+1* kez ve *n*++, *n* kez döndü. Döngünün içindeki değer atamalar ve if yapısı da *i++* ile aynı olacak şekilde *n* kez döndü. Toplamda zaman karmaşıklığı *T(n)= 3+1+n+(n+1)+3n+1* işlemleri yapılarak *T(n)=5n+6* olarak hesaplandı.



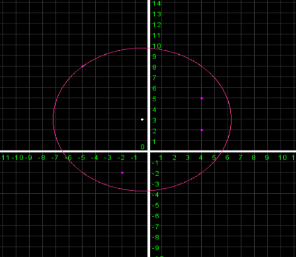
Kod Parçası 4

Son olarak *ucnoktasi\_verilencember* fonksiyonu hesapladık. Fonksiyonda hiç döngü bulunmadığı ve sadece atama değerlerinde oluştuğu için her bir atama değeri sadece bir kez döndü. Toplamda zaman karmaşıklığı *T(n)=22* olarak hesaplandı. Bütün değerleri bir araya getirdiğimizde algoritmanın zaman karmaşıklığı *T(n)=15n+27* olarak bulundu. Zaman karmaşıklığı üzerinden Big O notasyonunu hesapladığımızda ise *n* değerinin kuvvetinden *O(n)* değerini bulduk.

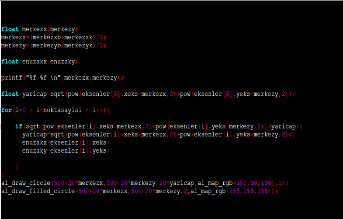
**III. Deneysel sonuçlar**

**III.I. Karşılaşılan problemler**

Çalışma süresinde ilk olarak denediğimiz algoritma en uzak *x* değerleri arasındaki mesafenin orta noktasını merkezin *x* değeri, en uzak *y* değerleri arasındaki uzaklığın ortasını merkezin *y* değeri olarak alıyordu. Kod Parçası 1 ve Kod Parçası 2’de ayrıntılı olarak incelenebilir. Algoritma sorunsuz bir şekilde ne kadar nokta girilirse girilsin hepsini çevreleyen bir çember çizdirebiliyordu fakat çemberin üzerinde birden fazla nokta olabileceğini hesaba katamadığı için bu çemberlere minimum demek mümkün değildi. Resim 3’te de görüldüğü gibi çember üzerine tek nokta alarak çalışan bir algoritmaydı. Daha sonra bu algoritmada kullandığımız *x* değerleri arasındaki mesafenin yarısını alma işlemini asıl kodumuzda kullandık.

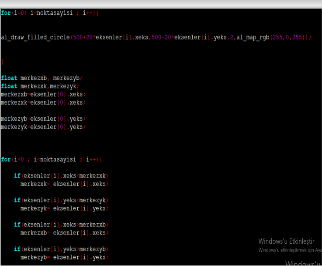


Resim

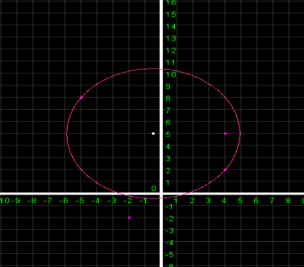


Kod Parçası 4: Algortima-1

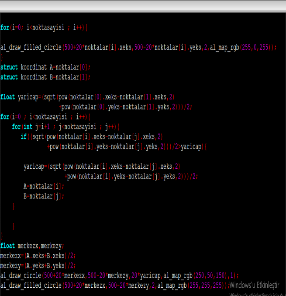
İkinci olarak uyguladığımız algoritmada ise *x* ve y değerleri arasındaki değil en uzak noktalar arasındaki mesafeyi kullandık ve en uzak olan iki nokta arasındaki mesafeyi çap olarak kabul ettik *(Kod Parçası 3)*. Algoritma en uzak iki nokta haricindeki noktaların merkeze olan uzaklığı yarıçap uzunluğundan büyük olmadığı sürece doğru çalıştı. Merkeze en uzak iki noktadan biri olacak kadar uzak olmayıp ama merkeze uzaklığı yarıçaptan büyük olan noktalarda Resim 4’te de görüldüğü gibi o noktayı çemberin içine almayarak hata verdi. Algoritma merkezin üzerinde en az iki nokta olması şartını sağlayabiliyordu ama üçüncü bir nokta da olabileceği olasılığını hesaba katmıyordu. Bu algoritma yanlış değil fakat eksik bir algoritmaydı. Daha sonra bu algoritmaya üçüncü bir nokta olma ihtimalini ekleyerek asıl kodumuzda kullandık.



Kod Parçası : Algortitma-1



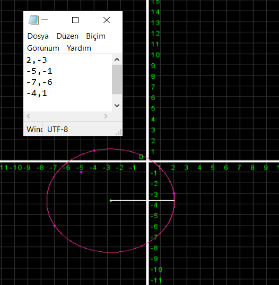
Resim

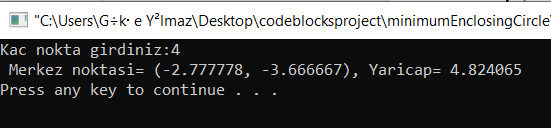


Kod Parçası 6: Algortima-2

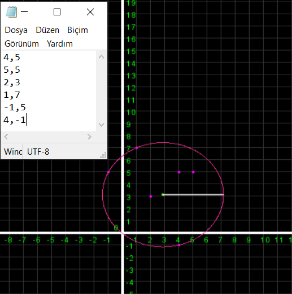
**III.II. Son algoritmanın çıktıları**

Son kısımda adım adım hatalarımızı düzelterek kodumuzu son haline ulaştırdık. Farklı değerler girerek çalışabilirliğini test ettik. Resim eklentilerinde de girilen değerler ve ekrana yansımaları gösterilmiştir.

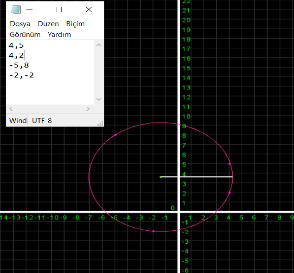


Resim 

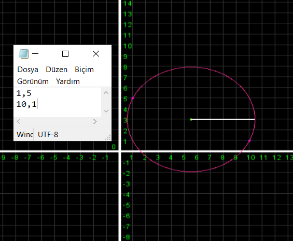
Resim



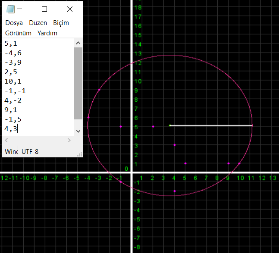
Resim

****

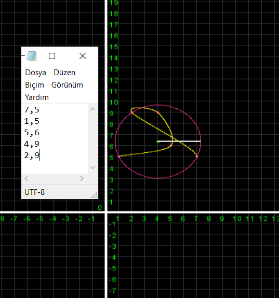
Resim



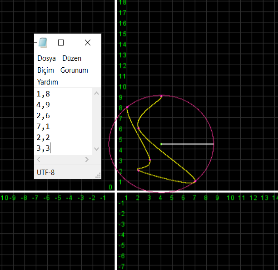
Resim



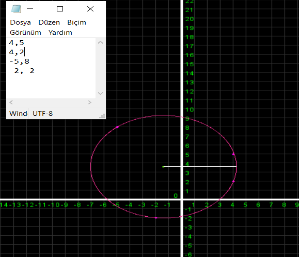
Resim

****

Resim 11



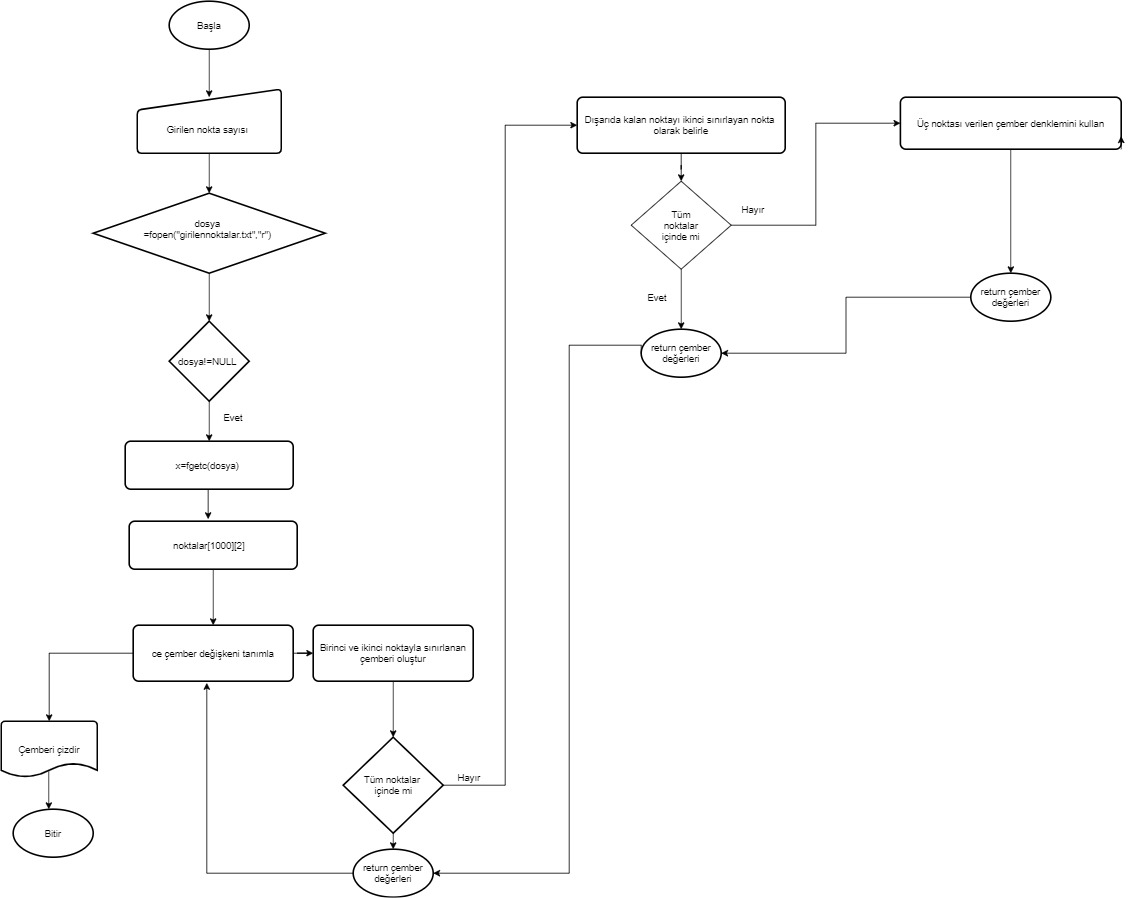
Resim 12

****

Resim 13

**IV. Sonuç**

Proje süresince bir problemi çözümlerken karşımıza ne tür hatalar çıkabileceğini ve bu hataları çalışma bütünlüğünü bozmadan nasıl düzeltebileceğimizi öğrendik. Bir projenin kodu yazılırken tek seferde ve tek bir yolla sonuca ulaşılmadığını, sürekli deneyerek ve yanılarak yöntemler arasında eleme yapıldığını deneyimlemiş olduk.



**V. Pseudo Kod**

1. Başla

2. Dosyayı aç

3. Dosyadan x değerlerini oku

4. Noktalar matrisi oluştur

5. Virgülden önceki kısmı noktalar[i][0]’e ata

6. Virgülden sonraki kısmı noktalar[i][1]’e ata

7. Noktaları cember\_bul fonksiyonuna gönder

8. Tüm noktaları kapsıyorsa çember değerlerini döndür kapsamıyorsa teknokta\_disarda fonksiyonuna gönder

9. Tüm noktaları kapsıyorsa çember değerlerini döndür kapsamıyorsa ikinokta\_disarda fonksiyonuna gönder

10. Tüm noktaları kapsıyorsa çember değerlerini döndür kapsamıyorsa ucnoktasi\_verilencember fonksiyonuna gönder

11. Gönderilen üç noktayı denkleme yerleştir çember değerlerini döndür

12. Döndürülen değerleri al\_draw\_circle eklentisi ile çember çizdirmede kullan

13. Merkez ve yarıçap değerlerini çıktı olarak yazdır

14. Nokta sayısı değişkeni ve p0,p1,p2,p3 kontrol noktalarıyla spline eğrisinin *x*, *y* değerlerini belirle

15. al\_draw\_pixel ile spline çizdir

16. Bitir

**VI. Kaynakça**

<https://en.m.wikipedia.org/wiki/Smallest-circle_problem>

<http://www.ambrsoft.com/trigocalc/circle3d.htm>

<https://liballeg.org/a5docs/trunk/primitives.html>

<https://www.geeksforgeeks.org/minimum-enclosing-circle-set-1/><http://capyayinlari.com.tr/demo/geometri-cember.pdf>

<https://liballeg.org/a5docs/trunk/font.html>

<https://en.wikipedia.org/wiki/Centripetal_Catmull%E2%80%93Rom_spline>

<http://cagataykiziltan.net/programin-calisma-hizi-ve-algoritma-verimliligi/zaman-karmasikligi-ve-buyuk-o-notasyonu-time-complexity-and-big-o-notation/>

<https://www.nayuki.io/res/smallest-enclosing-circle/computational-geometry-lecture-6.pdf>

<https://acikders.ankara.edu.tr/pluginfile.php/11954/mod_resource/content/0/FZM206Hafta03.pdf>

<http://bilgisayarkavramlari.sadievrenseker.com/2009/08/10/splines-seritler/>

<https://en.wikipedia.org/wiki/B-spline>

<https://en.wikipedia.org/wiki/Spline_(mathematics)>

<https://mathworld.wolfram.com/B-Spline.html>

<https://web.itu.edu.tr/yukselen/HM504/02Ek-%20Bezier%20e%F0rileri.pdf>

<https://www.allegro.cc/manual/5/graphics.html>

<http://www.abdullahsivari.com/wp-content/uploads/%C3%A7emberin-analitik-incelenmesi.pdf>

<http://www.sunshine2k.de/coding/java/Welzl/Welzl.html>

<https://www.cse.iitk.ac.in/users/ssahai/talks/sec.pdf>

<https://people.inf.ethz.ch/emo/PublFiles/SmallEnclDisk_LNCS555_91.pdf>