**Kocaeli Üniversitesi**

**Bilgisayar Mühendisliği Bölümü**

**Programlama Laboratuvarı I**

**Minimum çevreleyen çember ve B Spline Eğrisi**

*Oğuz Narlı Alperen Tan*

[*180201074@kocaeli.edu.tr*](mailto:180201074@kocaeli.edu.tr)[*190201054@kocaeli.edu.tr*](mailto:190201054@kocaeli.edu.tr)

***Projenin Özeti:***

Programlama Laboratuvarı I projesi olarak bizlerden “Minimum çevreleyen çember” ve “B spline eğrisi” algoritmalarını grafik arayüzünde gösteren bir uygulama yazmamız beklenmektedir.

Yazdığımız algoritma kullanıcının belirttiği dosya dizinindeki txt dosyasını okuyup, dosyanın içindeki koordinat değerlerine uygun, koordinat düzleminde minimum çember ve B spline eğrisi çiziyor.

***Giriş:***

Projede belirtilen dosya dizinindeki txt’deki bilgiler okunur,üzerinde matematiksel işlemler yapılarak grafik arayüzünde çıktı olarak verilir. “graphics.h” kütüphanesi Codeblocks üzerinde C++ dili seçildiği zaman sorunsuz çalıştığı için projenin tamamı C++ projesinde C dili ile yazılmıştır.

***Temel Bilgiler:***

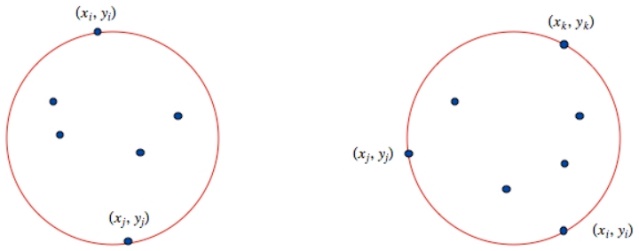
Proje C programlama dilinde geliştirilmiş olup IDE(geliştirme ortamı) olarak Codeblocks v.17.12, matematiksel işlemler için “math.h” kütüphanesi ve arayüz tasarım için “graphics.h” kütüphanesi kullanılmıştır.

Projede toplam 6 fonksiyon bulunmaktadır.

***Minimum Çember Giriş:***

Minimum çevreleyen çember problemi, içinde veya sınırında belirli bir nokta kümesi içeren en küçük yarıçaplı daireyi bulma problemidir. Bu daire minimum çevreleyen daire olarak bilinir. Problem ilk olarak 1857’de İngiliz matematikçi James Joseph Sylvester tarafından önerildi.

Problem aynı zamanda tesis konum problemine de örnek oluşturur. Tesis konum probleminde tesisin konumu müşterilerin seyahat edeceği uzaklığı minimuma indirgeyecek şekle uygun olarak seçilmelidir.



[1]Minimum çevreleyen çember

***Çember Hesaplama:***

Kullanıcının belirttiği dosya yolundaki txt dosyasından alınan {{x,y},{a,b}} formatındaki koordinat değerlerini diziye atadıktan sonra öncelikle dizinin boyutunu kontrol ederek kaç tane koordinat girildiğini hesapladık. Eğer bir koordinat var ise belirtilen noktayı en küçük çember olarak kabul ettik. 2 koordinat girildi ise girilen iki koordinat arasındaki uzaklığı çap kabul eden ve noktaların ortalamasını alarak oluşturduğumuz koordinatı merkez kabul eden bir çember çizdirdik. Girilen nokta sayısı 2’den fazla ise yazdığımız algoritma ile girilen bütün koordinat değerlerini birbiri arasında karşılaştırarak birbirine en uzak 2 koordinat değerini bulup, belirlenen koordinatlar arasındaki uzunluğu çap kabul eden ve koordinatların ortalamasını merkez kabul eden bir çember çizdirdik. Son olarak yazdığımız algoritma ile belirlenen en uzak 2 koordinat değerlerine en uzak 3. noktayı bulup noktanın belirlenen çemberin merkezine olan uzaklığının yarıçapın uzunluğundan büyük olup olmadığını kontrol ettik. Eğer uzun ise yukarıda belirtmiş olduğumuz algoritmalar minimum çember için yetersiz kalıyor anlamına geldiğinden matematiksel işlemler ile belirlenen birbirine en uzak 3 noktayı baz alan yeni bir merkez noktası belirleyip, bu noktadan en uzak herhangi 3 noktadan bir tanesine olan uzunluğu da yarıçap kabul ederek yeni bir çember çizdirdik.

3 noktası olan çember çizimi için kullandığımız algoritmayı geometri kullanarak sağladık. Geometride üçgenin kenar ortadikmesinin kesiştiği nokta kendisini çevreleyen çemberin merkezine tekabül etmektedir. Önce birbirlerine herhangi 2 kenar çizdik ve normalinin eğimini

n=x1-x2/y2-y1 şeklinde aldık. Normalin denklemi nx+c=y dir.2 kenarın kendi ortalarının kenarlarını aldık ve denklemi yerleştirdik. Böylece 2 tane normalin denklemi olacaktır. Sonra bunları eşitleyerek çember merkezini bulduk. Çember merkezinden 3 tane noktadan herhangi birine olan uzaklığı hesaplayarak yarıçapı da bulduk.

Yazdığımız algoritmalar sonucu ortaya çıkan çember, koordinat düzlemindeki bütün noktaları kapsıyor ve mümkün olan en küçük çemberi çizdi.

***Minimum Çember Kaba Kod:***

for(i=0,i<size,i++)

print(bütün koordinatlar);

En uzak 2 nokta tanımla(x1,y1,x2,y2);

if(girilen koordinat sayısı=1)

merkez x=x1 merkezy=y1

else if(girilen koordinat sayısı=2)

merkez x=(x1+x2)/2 merkez y=(y1+y2)/2

yaricap=√(x1-merkezx)² + (y1-merkezy)²

else

int en büyük uzunluk=0

for(i=0’dan size’a kadar i++)

for(j=i+1’den size’a kadar j++)

if(√(pt[j].x-pt[i].x)² + (pt[j].y-pt[i].x)²>enbuyukuzunluk)

enbuyukuzunluk=√(pt[j].x-pt[i].x)² + (pt[j].y-pt[i].x)²

x1=pt[i].x; y1=pt[i].y; x2=pt[j].x; y2=pt[j].y;

int en uzak 3. uzunluk=0,x3,y3;

for(i=0’dan size’a kadar i++)

if(√(pt[i].x-x1)² + (pt[i].y-y1)² + √(pt[i].x-x2)² + (pt[i].y-y2)²>enuzakucuncuuzunluk)

en uzak üçüncü uzunluk=√(pt[i].x-x1)² + (pt[i].y-y1)² + √(pt[i].x-x2)² + (pt[i].y-y2)²

x3=pt[i].x; y3=pt[i].y;

print(x1,y1 x2,y2’ye en uzak 3. Nokta x3,y3)

merkez\_x=(x2+x)/2; merkez y=(y2+y)/2;

if(en uzak 3. noktanin merkeze uzakligi > yarıçap)

yeni çember merkezi ve yarıçap hesapla;

print(çemberin merkez koordinatı ve yarıçapı);

else

print(çemberin merkez koordinatı ve yarıçapı);

***Minimum Çevreleyen Çember Kod İçeriği:***

for(int i=0;i<size;i++) {

printf("( %d , %d )",(pt[i].x-250)/10,(250-pt[i].y)/10); }

Öncelikle yukarıdaki kod parçası ile ekrana kullanıcının yolunu belirttiği txt dosyasından alınan koordinatları graphics.h’ın ana koordinat değerlerine göre uyarlayarak ekrana yazdırdık. Bunun için koordinatları 250 çıkarma veya 250 ekleme ve 10’a bölme işlemlerinden geçirdik. Yapılan işlemler koordinat değerinin x veya y düzleminde olmasına göre değişiyor.

float x,y,x2,y2;

float mer\_x=0;

float mer\_y=0;

float yaricap=0;

if(size==1) {

printf("Cemberin merkezi=(%d %d)\nCemberin yaricapi=0(nokta)\n",(pt[0].x-250)/10,(250-pt[0].y)/10); }

Ardından çemberi belirlemek için kullanacağımız değişkenleri belirleyip, girilen koordinat sayısını kontrol ettikten sonra girilen koordinat sayısı 1 ise belirlenen koordinattaki noktayı çember kabul ederek ekrana yazdırdık.

else if(size==2) {

mer\_x=(pt[0].x+pt[1].x)/2.0;

mer\_y=(pt[0].y+pt[1].y)/2.0;

yaricap=sqrt(pow(pt[1].x-mer\_x,2)+pow(pt[1].y-mer\_y,2));

printf("Cemberin merkezi=(%.2f %f)\nCemberin yaricapi=%.2f\n",(mer\_x-250)/10,(250-mer\_y)/10,yaricap/10); }

Eğer girilen koordinat sayısı 2 ise belirtilen 2 noktanın ortalamasını merkez koordinat kabul edip aralarındaki uzaklığı yarıçap olarak alarak ekrana yazdırdık.

else {

float ebu=sqrt(pow(pt[1].x-pt[0].x,2)+pow(pt[1].y-pt[0].y,2));

for(int i=0;i<size;i++) {

for(int j=i+1;j<size;j++) {

if(sqrt(pow(pt[j].x-pt[i].x,2)+pow(pt[j].y-pt[i].y,2))>ebu){

ebu=sqrt(pow(pt[j].x-pt[i].x,2)+pow(pt[j].y-pt[i].y,2));

x=pt[i].x;

y=pt[i].y;

x2=pt[j].x;

y2=pt[j].y; } } }

printf("Iki nokta arasi en buyuk uzaklik=%.2f\nEn uzak noktalarin koordinatlari=(%.2f,%.2f)-(%.2f,%.2f)\n",ebu/10,(x-250)/10,(250-y)/10,(x2-250)/10,(250-y2)/10);

Eğer girilen koordinat sayısı 2’den büyük ise öncelikli olarak karşılaştırma yapmak için bir en büyük uzunluk belirledik.

Ardından girilen koordinat değerlerini yazdığımız 2 for döngüsü ile karşılaştırıp, birbirlerine en uzak 2 koordinat değerini ve aralarındaki uzaklığı hesaplayıp ekrana yazdırdık.

float x3,y3,enuzakucuncuuzunluk=0;

for(int i=0;i<size;i++) {

if(sqrt(pow(pt[i].x-x,2)+pow(pt[i].y-y,2))+sqrt(pow(pt[i].x-x2,2)+pow(pt[i].y-y2,2))>enuzakucuncuuzunluk){

enuzakucuncuuzunluk=sqrt(pow(pt[i].x-x,2)+pow(pt[i].y-y,2))+sqrt(pow(pt[i].x-x2,2)+pow(pt[i].y-y2,2));

x3=pt[i].x;

y3=pt[i].y; } }

printf("(%.2f,%.2f)-(%.2f,%.2f)'ya en uzak ucuncu nokta=(%.2f,%.2f)\n",(x-250)/10,(250-y)/10,(x2-250)/10,(250-y2)/10,(x3-250)/10,(250-y3)/10);

Yukarıda en uzak üçüncü uzaklık ve en uzak 3. koordinat değerimiz için değişkenlerimizi belirleyip noktaların önceden belirlediğimiz 1. ve 2. noktaya olan uzaklığını hesaplayarak en uzak 3. noktamızı belirleyip ekrana yazdırdık.

mer\_x=(x2+x)/2;

mer\_y=(y2+y)/2;

yaricap=sqrt(pow(x-mer\_x,2)+pow(y-mer\_y,2));

En uzak 2 noktaya göre bir merkez koordinatı ve yarıçap uzunluğu belirledik.

if((sqrt(pow(x3-mer\_x,2)+pow(y3-mer\_y,2)))>yaricap) {

float c23=0;

float c12=0;

float mb\_norm=0;

float ma\_norm=0;

c23=((y2+y3)/2)-(((x2+x3)/2)\*((x3-x2)/(y2-y3)));

c12=((y+y2)/2)-(((x+x2)/2)\*((x-x2)/(y2-y)));

ma\_norm=((x-x2)/(y2-y));

mb\_norm=((x3-x2)/(y2-y3));

mer\_x=(c23-c12)/(ma\_norm-mb\_norm);

mer\_y=(ma\_norm\*mer\_x)+c12;

yaricap=sqrt(pow(x-mer\_x,2)+pow(y-mer\_y,2));

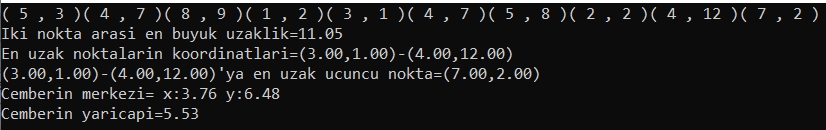
printf("Cemberin merkezi= x:%.2f y:%.2f\nCemberin yaricapi=%.2f\n",(mer\_x-250)/10,(250-mer\_y)/10,yaricap/10); }

Belirlenen en uzak 3. koordinat değerinin merkeze olan uzaklığı ile yarıçapın uzunluğunu karşılaştırarak 3. noktanın dışarıda kalıp kalmadığını kontrol ettik. Eğer merkeze olan uzaklık yarıçaptan daha büyükse, bu noktanın çizilecek olan çemberin dışında kalacağını gösterdiğinden matematiksel işlemler ile 3 koordinat değerine de uygun yeni bir merkez noktası oluşturup çemberimizi oluşturup ekrana yazdırdık.

else {

printf("Cemberin merkezi= x:%.2f y:%f\nCemberin yaricapi=%.2f\n",(mer\_x-250)/10,(250-mer\_y)/10,yaricap/10); } }

En uzak 3. koordinat değerinin merkeze olan uzaklığı yarıçaptan küçükse bu durum 3. en uzak noktanın çizilecek çemberin içinde kalacağını gösterdiğinden çemberimizi önceki en uzak 1. ve 2. nokta arasında belirlenen merkez koordinatlarına uygun olduğunu belirleyip merkez koordinatlarını ekrana yazdırdık.



Ekrana yazdırılan çıktı

***Grafik Arayüzü Kod İçeriği:***

initwindow(600,600);

settextstyle(SMALL\_FONT,HORIZ\_DIR,4);

char merkez[50];

sprintf(merkez,"Merkez=(%.2f,%.2f)",(mer\_x-250)/10,(250-mer\_y)/10);

char yaricap\_[50];

sprintf(yaricap\_,"Yaricap=(%.2f)",yaricap/10);

outtextxy(0,0,merkez);

outtextxy(0,10,yaricap\_);

initwindow fonksiyonu ile açacağımız grafik arayüzü penceresinin boyutunu belirledik, settextstyle ile yazı tipimizin şeklini ve boyutunu ayarladık. Grafik arayüzünün sol üstünde merkez ve yarıçap değerlerini yazıp göstermek için outtext fonksiyonunu kullandık. Kullandığımız grafik arayüzü, başlangıç koordinatını ekranın sol üstü olarak belirlediği için başlangıç koordinatları olarak (x,y) 0,0 ve 0,10 kullandık. outtext’in içine yazdığımız merkez ve yarıçap değişkeninin dinamik olması için (kullanıcının girdiği veriye göre ekrana yazması için) char tipinde bir dizi belirleyip dizinin içine yarıçap değerini atadık. Son olarak outtextxy fonksiyonu ile ekranın sol üstüne belirlediğimiz değerleri yazdırdık.

moveto(250,250);

lineto(250,0);

lineto(250,500);

moveto(250,250);

lineto(500,250);

lineto(0,250);

moveto(0,0);

Yukarıdaki moveto fonksiyonu ile çizdireceğimiz koordinat ekseninin orijinini 250,250 koordinatı olarak belirledik. lineto fonksiyonları ile koordinat düzleminin x ve y eksenlerini çizdirdik.

outtextxy(250,250,"0");

outtextxy(500,255,"25");

outtextxy(235,0,"25");

…

Arka arkaya yazdığımız outtextxy fonksiyonları ile x ve y ekseninin üstüne koordinat sayılarını ekledik. 0’dan 25’e kadar yazmanın yeterli olduğunu düşünüp, ekrana verilen çıktının çok yüksek sayılardan oluşmaması için normal değerleri 10’a bölmenin yeterli olduğunu karar kılarak ekrana verilen çıktıları buna göre ayarladık. Aşağıdaki kod dizinlerinde de bu örnekleri görebilirsiniz.

circle(mer\_x,mer\_y,yaricap);

Çember algoritmamızda belirlenen merkez koordinatları ve yarıçap değerine göre circle fonksiyonu ile çemberimizi çizdirdik. moveto(mer\_x,mer\_y);

lineto(x2,y2);

Yukarıdaki kod dizisini kullanarak moveto ile başlangıç çizim koordinatını çemberin merkezine taşıyıp lineto ile çember üzerindeki 2 noktadan biri olan en uzak 2. noktayı referans alıp çizgi çizerek yarıçapı çemberin içinde gösterdik.

setcolor(9);

for(int i=0;i<size;i++) {

circle(pt[i].x,pt[i].y,2);

floodfill(pt[i].x,pt[i].y,9);

char koord[50];

sprintf(koord,"(%d,%d)",(pt[i].x-250)/10,(250-pt[i].y)/10);

outtextxy(2+pt[i].x,pt[i].y-2,koord); }

Sırada kullanıcıdan alınan bütün koordinat değerlerini grafik arayüzünde x ve y koordinat düzlemi üzerinde göstermek vardı. Çizilecek noktaların ve koordinat değerlerinin görünürlüğünü arttırmak için setcolor fonksiyonunu kullandık. Noktaları çizmek için for döngüsü içinde, piksel çizdirmek yerine circle fonksiyonunu kullandık. Piksel, arayüzümüzde çok küçük bir yer kapladığı için görünürlük açısından beklediğimiz sonucu veremiyordu. floodfill fonksiyonu ile çizilen noktaların içini beyaza boyadıktan sonra noktaların koordinatlarını dinamik olarak yazdırmak için koord isminde bir dizi oluşturup sprintf ile içine koordinat değerlerini koyduktan sonra outtextxy fonksiyonu ile grafik ekranına yazdırdık.

circle(mer\_x,mer\_y,2);

floodfill(mer\_x,mer\_y,9);

char merkezkoord[50];

sprintf(merkezkoord,"(%.1f,%.1f)",(mer\_x-250)/10,(250-mer\_y)/10);

outtextxy(mer\_x-37,mer\_y-10,merkezkoord);

draw(size,pt);

getch();

closegraph();

Son olarak circle fonksiyonunu çemberin merkezini nokta ile belirtmek için kullandıktan sonra tekrar floodfill ile merkez noktasını boyayıp üstüne outtextxy ile koordinat değerlerini yazdırdık. getch fonksiyonu ile grafik arayüzünün hemen kapanmasını engelleyip closegraph fonksiyonu ile yazdığımız kodu sonlandırdık.

***Çember İçin Karmaşıklık Hesabı:***

Çember ile ilgili bütün işlemler main fonksiyonunda bulunduğu için main fonksiyonu dikkate alınarak hesaplama yapılacaktır.

Ekrana yazdırmak için kullandığımız olan ilk for döngüsü *0*’dan *n*’e kadar döndüğünü varsayarak karmaşıklığına *n* diyebiliriz. Diğer işlemlerin sayısı çok döngülerin yanında çok küçük kaldığı için dikkate almadık.

Diğer for döngüleri durumlara bağlı olduğu için:

Eğer girilen koordinat sayısı 1 ise toplam karmaşıklık *n*;

Girilen koordinat sayısı 2 ise başka bir döngü gerçekleşmediği için karmaşıklık yine *n*;

Girilen koordinat sayısı 2’den fazla ise,

for(int i=0;i<size;i++)

for(int j=i+1;j<size;j++)

döngüsü devreye girdiğinden ve bu döngünün karmaşıklığı *(n-1)+(n-2)+(n-3)…+(1)=(n²-n)/2* olduğundan toplam karmaşıklığa *(n²-n)/2* eklenir.

Ardından gelen for döngüsü ilk döngüyle aynı sayıda döneceğinden karmaşıklığa *n* eklenir.

Yani girilen koordinat sayısı 1 ise karmaşıklık *n+n*, 2 ise karmaşıklık yine *n+n*, 2’den fazla koordinat girilir ise karmaşıklığımız: *n+n+(n²-n)/2+n=3n+(n²-n)/2* olur. Karmaşıklığa etkisi az olan değerleri çıkardığımız zaman:

Girilen koordinat sayısı bir ise *O(n)*

Girilen koordinat sayısı iki ise *O(n)*

Girilen koordinat sayısı ikiden fazla ise *O(n²)* diyebiliriz.

***Çember Sonuç:***

Yapılan işlemler sonucunda kullanıcının belirttiği dosya yolundaki txt dosyasından koordinat okuma işlemi sorunsuz bir şekilde gerçekleştirildi. Okunan koordinat değerleri karşılaştırılarak bir çember merkezi ve yarıçapı belirlendi. Belirlenen çemberin bütün koordinat değerlerindeki noktaları kapsaması sağlandı. Ardından yazılan grafik arayüzü ile önceden belirlenen çember merkezine ve yarıçapa uygun olarak bir çember çizildi, çizilen çemberin koordinat düzleminde bulunan bütün noktaları kapsadığını teyit ettik. Çıktıyı “Deneysel Sonuç” bölümünde görebilirsiniz.

***B-Spline***

***B-Spline Giriş:***

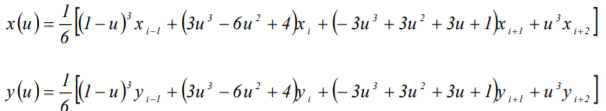
Veri uydurma, geometrik modelleme (Hoschek, Lasser vd., 1993, de Boor 2001, Piegl ve Tiller 2012) ve bilgisayar destekli tasarım / imalat (CAD / CAM) gibi alanlarda çok önemli bir araştırma konusudur (Barnhill, 1992, Pottmann vd., 2005, Patrikalakis ve Maekawa 2009). Fiziksel bir nesnenin taranması ile elde edilen nokta bulutundan tekrar nesnenin dijital bir modelini oluşturmak için tersine mühendislik kullanılır. Bu yüzden tersine mühendislik veri uydurma probleminde temel bir araç haline gelmiştir (Varady vd., 1997, Farin vd., 2002, Gálvez ve Iglesias 2012, Gálvez vd., 2012). Çoğu zaman, verilerin altında yatan fonksiyon karmaşıktır ve bu nedenle basit bir fonksiyonla bu verilerin temsili zordur (Gálvez ve Iglesias 2011). Bu gibi durumlarda, çok esnek, yaygın olarak bulunabilen, güçlü matematiksel özelliklere (yerel modifikasyon, projektif değişmezlik, güçlü dışbükey gövde vb.) sahip oldukları ve çok çeşitli şekilleri temsil edebildikleri için B-spline’lar en çok tercih edilen yaklaşım fonksiyonlarıdır (Farin 2002, Yuan vd., 2013). Bilgisayar Destekli Tasarım (CAD) ve Bilgisayar Destekli Mühendislik (CAE), B-spline’lar ve bunların genelleştirilmiş formları NURBS (Ma ve Kruth, 1995, Ma ve Kruth, 1998, de Boor, 2001, Piegl ve Tiller, 2012), nesnenin şeklini/geometrisini tasarlamak ya da ölçüm verilerinden fiziksel parçaların geometrik modellerini yeniden oluşturmak için kullanılır. Bspline'lar ayrıca sinyal işleme ve görüntü işleme gibi alanlarda genellikle gürültülü sinyalleri işlemek veya karmaşık fonksiyonları yaklaştırmak için kullanılır (Unser vd., 1993, Precioso, Barlaud vd., 2003).

***B-Spline Matematiksel Denklemleri:***

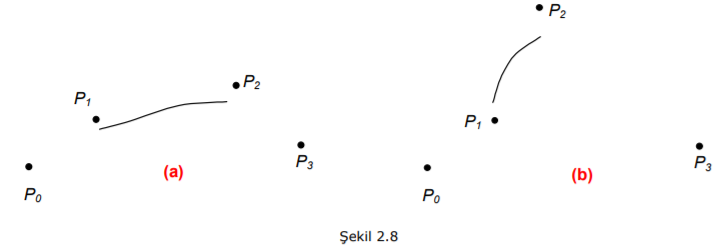
B-splayn eğrileri de Bezier eğrileri gibi veri noktalarından geçmeyen eğrilerdir. (Bu açıdan en küçük kareler eğrilerine benzerler). Herhangi bir derecede olabilirler. Ancak burada kübik eğrilerle ilgilenilecektir. Kübik B-splayn eğrileri iki nokta arasından geçirilen basit kübik eğrilere benzerdir. Ancak bu eğriler veri noktalarından geçmek zorunda değildir.

N+1 adet nokta Pi (xi , yi), i = 0,1,2,...,N şeklinde verilmiş olsun. Herhangi bir Pi-Pi+1 (i = 0,1,2,...,N) aralığındaki kübik B-splayn eğrisi parametrik olarak

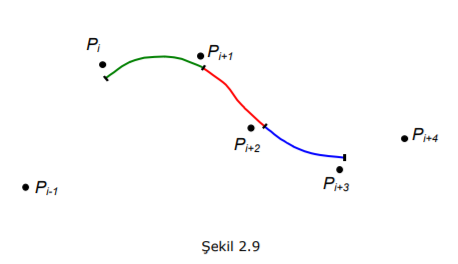
şeklindedir. Buradaki bk katsayıları dört noktanın etkisi için ağırlık faktörleri olup

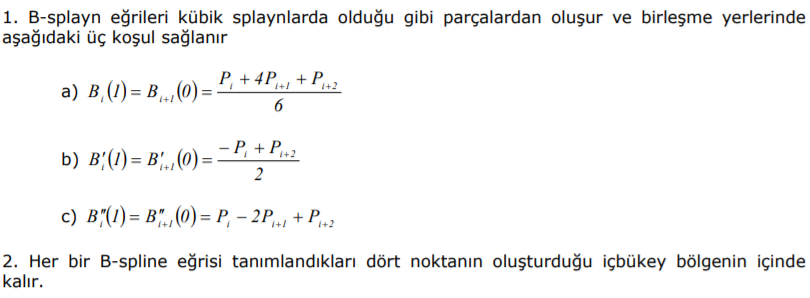
şeklinde tanımlanmaktadır. Bi yerine sırasıyla x ve y, Pi yerine de xi ve yi konularak eğri üzerindeki noktaların koordinatları

şeklinde elde edilecektir. Noktaların ağırlık faktörleri u parametresi ile değişmekte olup u=0 için sırasıyla 1/6, 2/3, 1/6, 0 ve u=1 için de 0,1/6, 2/3, 1/6 dır.

Şekil 2.8 a-b de dört nokta ile belirlenmiş bir B-splayn eğrisinin noktalardan sadece birinin yer değiştirmesi halindeki durumu göstermektedir. P2 noktası önce yukarı ve daha sonra sola doğru çekilince, bekleneceği gibi eğri de bunu izleme eğiliminde olup P1 noktasının ötesine geçmiştir. Eğrinin başladığı ve bittiği bu iki noktaya da çok yakın olmadığı ilgi çekicidir. Bu aralıktaki eğrinin tanımlanması için P0, P1, P2 ve P3 gibi dört nokta kullanılmıştır.

Bir B-spline eğrini tanımlamak için dört noktaya ihtiyaç vardır. Daha fazla sayıdaki noktadan Bsplayn eğrileri geçirmek için kübik splayn uygulamalarındakine benzer bir yöntem kullanılabilir. Bu durumda eğrinin sürekliliği için koşullar alelade splaynlardakine benzer şekilde birinci ve ikinci türevlerinin sürekliliği koşullarıdır. Ağırlık faktörleri için yukarıda verilen bağıntılar bu koşullar sağlanacak şekilde çıkartılmıştır. Şekil 2.9 da Bir B-splayn eğrinin ardarda üç parçası görülmektedir.



B-splayn eğrilerinin özellikleri şu şekilde özetlenebilir:

***B Spline Kaba Kod:***

Girdiler:koordinat[],boyut

pt[4] ,splinex spliney tanımla

pt[0]=0;pt[1]=koordinat[0];

pt[2]=koordinat[1];pt[3]=koordinat[2];

for i->3’ten boyuta kadar:

pt[0]=pt[1];pt[1]=pt[2];pt[2]=pt[3];

pt[3]=koordinat[i];

int uzaklık=pt[1] ile pt[2] arasındaki uzaklık

for j->0’ dan uzaklığa kadar:

float u=j/uzaklık

splinex[j]=

spliney[j]=

for j->0’ uzaklığa kadar:

çizdir(splinex[j],splinex[jy])

***B Spline Kod İçeriği:***

double splinex[1000];

double spliney[1000];

Global olarak tanımlanmış bu iki dizi amacı eğrinin izleyeceği koordinatları içermektedir. “splinex” x ,“spliney” y koordinatlarını tutmaktadır.

void bspl(struct point pt1,struct point pt2,struct point pt3,struct point pt4,int u){

for(int i =0;i<u;i++){

float division=(float)i/(float)u;

splinex[i]=(((pow(1-division,3)\*pt1.x)+((3\*pow(division,3)-6\*pow(division,2)+4)\*pt2.x)+((-3\*pow(division,3)+3\*pow(division,2)+3\*division+1)\*pt3.x)+(pow(division,3)\*pt4.x)))/6;

spliney[i]=(((pow(1-division,3)\*pt1.y)+((3\*pow(division,3)-6\*pow(division,2)+4)\*pt2.y)+((-3\*pow(division,3)+3\*pow(division,2)+3\*division+1)\*pt3.y)+(pow(division,3)\*pt4.y)))/6;

} }

Yukarıdaki fonksiyonumuz eğriyi hesaplayıp bulduğu koordinatları yukarıdaki kod parçacığında bulunan dizilerin dizinlerine eklemektedir. Bu fonksiyon 5 parametre almaktadır.4’ü nokta 5. si arasındaki eğriyi hesaplamak istediğimiz 2 noktanın arasındaki uzaklıktır. 2. ve 3. parametreler aralarındaki eğriliği hesaplamak istediğimiz noktalar ,1. ve 4. parametre eğriliği hesaplamak için geçtiğimiz bölümdeki denklemlerde bahsedildiği gibi gereken bir önceki ve sonraki noktalardır. 5. parametre denklemde girilen koordinatları 0 ile 1 arasına daraltmak ve eğriliği hesaplamak için döngü sayısını temsil eder. “splinex[i]=” ve “spliney[i]=” ifadelerinin eşitlendiği kod parçacıkları geçtiğimiz bölümde bahsettiğimiz denkleme göre yazılmıştır.

void draw(int size,struct point \*pt){

struct point pnt[4];

pnt[0].x=0;

pnt[0].y=0;

pnt[1]=pt[0];

pnt[2]=pt[1];

pnt[3]=pt[2];

for(int i =3;i<size;i++){

pnt[0]=pnt[1];pnt[1]=pnt[2];pnt[2]=pnt[3];pnt[3]=pt[i];

double interpolation=sqrt(pow(pnt[2].x-pnt[1].x,2)+pow(pnt[2].y-pnt[1].y,2));

int len =(int)interpolation;

bspl(pnt[0],pnt[1],pnt[2],pnt[3],temp);

for(int j =0;j<len-1;j++){

putpixel(splinex[j],spliney[j],BLUE);

}

}

}

Yukarıdaki fonksiyonumuz 1 önceki kod parçacığındaki fonksiyonu çağırıp oradan elde ettiğimiz koordinatları ekrana çizdirmekle görevlidir. Koordinat sayısını alan size ve koordinatları içeren pt olmak üzere 2 farklı parametre almaktadır. “pnt[4]” dizisi “bspln” yani eğriyi hesaplayacak 4 koordinatı içeren dizidir.

Hemen altındaki ifadeler, dosyadan alığımız koordinatların ilk 3’ ünü eşitleyerek başlangıç değeri olarak atadığımız kod parçacıklarıdır.

Daha sonra bu ifadeler altındaki for döngüsü içerisinde hemen bir önceki dizinine en sonuncusu da dosyadan aldığımız yeni koordinata eşitlenir ve yeni eğri hesaplamak için yeni ikili nokta elde etmiş oluruz.“pnt[1]” ve “pnt[2]” arasındaki eğriliği hesaplayacağımız 2 noktadır. Diğer iki nokta yani “pnt[0]” ve “pnt[3]” bir önce gelen ve sonra gelen noktaları temsil eder.“intepolation” değişkeni ise iki nokta nokta arası uzaklıktır. Sonra “len” değişkenine tamsayıya dönüştürülerek hemen altında çağırılan “bspl” fonksiyonuna parametre olarak gönderilir. En sonda yer alan for döngüsünde “putpixel” fonksiyonuyla noktalar çizdirilir.

***B-spline İçin Karmaşıklık Hesabı:***

B-spline ile ilgili bütün işlemler draw fonksiyonunda toplanmaktadır. Bu yüzden draw fonksiyonu üzerinde hesap yapılacaktır. İlk for döngümüze kadar 6 tane işlem gerçeklemektedir fakat bu sayı esas karmaşıklığın yanında küçük sayı olduğundan göz ardı ediyoruz. İlk for döngüsünü i=3 ten n ye kadar döndüğünü var sayalım bu yüzden n-3 kez dönücektir.2. ve 3. for döngüleri ilk for un içide olan döngülerdir.2. for işlemi bspl fonksiyonuda yer almaktadır.2. for kadar 6 tane işlem vardır döngü boyunca (n-3)\*6 tane işlem gerçekleşecektir. 2. j=0’ dan for unda k ye kadar işlem yapılmış olsun. Dolayısıyla k ,döngü içinde olduğundan ve içerisinde 3 işlem yaptığından 3k\*(n-3) kez işlem yapacaktır. 3. döngü içinse ise j=0’dan L’ye kadar işlem yapsın. İçerisinde tek işlem gerçekleştiğinde o döngü için toplam işlem sayısı (n-3)\*L kadardır.Hepsini topladığımızda ve işleme etki etmeyen sayıları çıkardığımızda;

f(n,k,L)=6n+3kn-3k+Ln-3L dir.

***B Spline Sonuç:***

txt dosyasından aldığımız koordinatlar başarılı bir şekilde “bspl” fonksiyonunda eğrili almış ve splinex ve spliney dizilerine izleyeceği koordinatlar başarılı şekilde dizinlerine eklenmiş “draw” fonksiyonuyla çizilmiştir. Çıktıyı “Deneysel Sonuç” bölümünde görebilirsiniz.

***Dosya İşlemleri:***

Not:Öncelikle koordinatlar dosyada “{{x1,y1},{x2,y2}} ” şeklinde saklanmaktadır.

int get\_size() {

FILE \*f=fopen(file\_path,"r");

//dosya

if(f!=NULL) {

int i=0;

while(!feof(f)) {

if(fgetc(f)==',') {

i++; } }

fclose(f);

return i; }

else {

printf("File does not exist"); } }

Yukarıda yer alan fonksiyon dosyadaki koordinatların sayısını belirlemektedir.“fgetc()” fonksiyonu ile karakter karakter okuyup aldığı karakterler de her ‘,’ karakterini gördüğünde boyut sayısını arttırmaktadır(int i=0;). En son “!feof(f)” yapısı false döndüğünde döngü biter ve i’yi boyut olarak döndürür.

Bu fonksiyonda mainde yer alana size değişkenine get\_size()/2+1 şeklinde atanır.“/2+1” ifadesi noktayı structer dizisi olarak tuttuğumuz için toplam koordinat sayısı yarısının 1 fazlasına tekabül etmektedir.

void readfile(char \*coordinate) {

FILE \*f=fopen(file\_path,"r");

int i=0;

if(f!=NULL) {

while(!feof(f)) {

char c;

c=fgetc(f);

if(c!='{'&&c!='}') {

coordinate[i]=c;

i++; } }

fclose(f); }

else {

printf("File does not exist"); } }

Şimdiki fonksiyonumuz dosyadan veri okuma görevini üstlenmektedir. Aldığı parametre karakter karakter okuduğumuz koordinatları string olarak bellekte saklamamızı sağlar. “fgetc(f);” fonksiyonuyla dosyadan karakter okuyup “char c;” değişkenine atamaktayız. Sonra if yapısıyla sayı olup olmadığı tespit edilir eğer sayı ise parametre olarak gelen stringin bir sonraki dizinine atanır. Mainde tanımladığımız “char coordinate[size\*5];” değişkenimiz “char \*coordinate” parametresine gönderilerek koordinatları string olarak içerir.

void parse\_coordinate(char \*coor,struct point pt[],int size) {

char \*say;

say=strtok(coor," ,");

pt[0].x=250+(((float)atoi(say))\*10);

say = strtok (NULL, " ,");

pt[0].y=250-(((float)atoi(say))\*10);

for (int i = 1; i < size; i++) {

say = strtok (NULL, " ,");

pt[i].x=250+(((float)atoi(say))\*10);

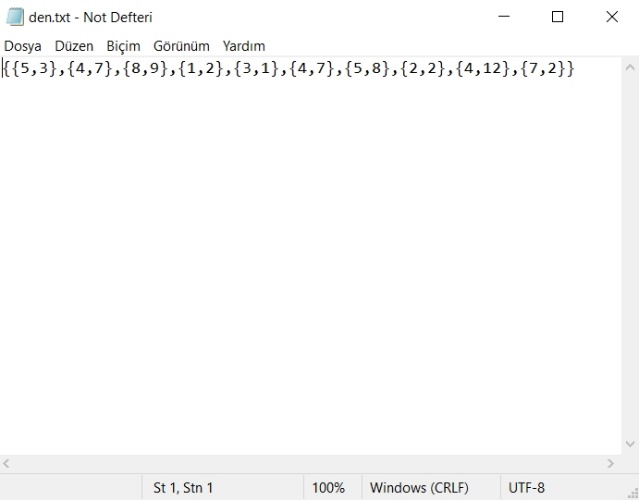
say = strtok (NULL, " ,");

pt[i].y=250-(((float)atoi(say))\*10);

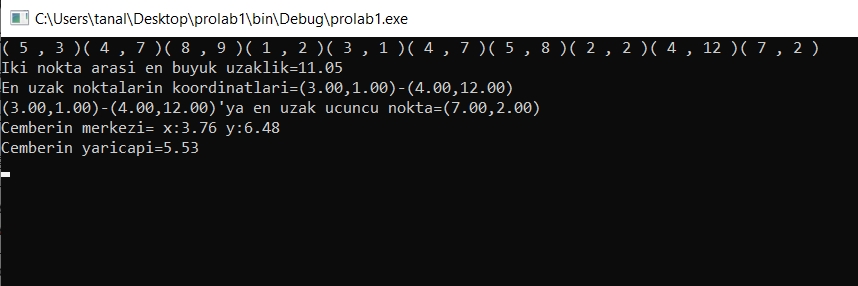
} }

Yukarıda belirttiğimiz fonksiyonumuz ise “char coordinate[size\*5];” değişkeninde yer alan koordinatları “strtok(coor," ,");” fonksiyonu ile ayırıp “(float)atoi(say)” sayıya çevirip main de tanımlanan “struct point pt[size];” dizisinin dizinlerine ekleme görevini üstlenmektedir.

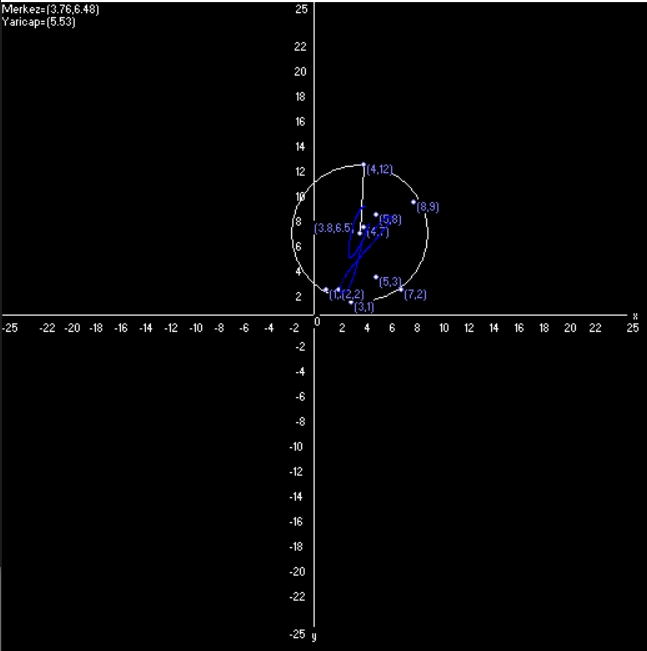
***Deneysel Sonuçlar:***



Not defterine girilen koordinat değerleri



Sonuç ekranı

 Grafik arayüzünde gösterilen çıktı

***Kaynakça:***

[1]<https://www.cs.princeton.edu/courses/archive/spr10/cos226/assignments/sec1.png>

<https://fatihalparslann.wordpress.com/2017/02/02/c-dilinde-graphics-h-kutuphanesi-1-bolum/>

https://en.wikipedia.org/wiki/Smallest-circle\_problem

<https://dergipark.org.tr/tr/download/article-file/571214>

<https://web.itu.edu.tr/yukselen/HM504/02Ek-%20Bezier%20e%f0rileri.pdf>

embedded.kocaeli.edu.tr

edestek.kocaeli.edu.tr