KOCAELİ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

PROGRAMLAMA LABORATUVARI 1

MİNİMUM ENCLOSİNG CİRCLE AND B-SPLİNE

FATMA GÜL YILDIRIM -->190201017

Okul Mail Adresi--> 190201017@kocaeli.edu.tr

KEREM KARATAŞ-->190201076

Okul Mail Adresi-->190201076@kocaeli.edu.tr

PROJENİN ÖZETİ:

İstenenler:

- ☺ 2 boyutlu bir düzlemde N nokta verildiğinde,bütün noktaları kapsayan (sınırından da geçebilir) bir minimum yarıçaplı çevreleyen çember istenmektedir. Oluşan bu çemberin yarıçapı ve merkezinin konsolda veya grafik ekranında gözükmesi istenmektedir.
- ☺ Verilen N noktanın en yakınından veya üstünden geçen eğriyi(B-Spline) çizdirmemiz gerekmektedir.
- ☺ Tüm bunların C programlama dili kullanılarak gerçekleştirilmesi istenmektedir.

1. Giriş:

- ☺ Projeyi C programla dilinde OpenGL arayüz kütüphanesini kullanarak yaptık.
- ☺ C programlama dili orta seviyeli dillere göre daha anlaşılır çok yüksek seviyeli dillere göre daha esnek bir dil olması sebebiyle orta segmentli bir dil diyebiliriz.Donanımı hitabeti güçlüdür.İşletim sistemi, aygıt sürücüsü vb şeyler yazılabilir.
- ☺ OpenGL kütüphanesi gelişmiş donanım desteğiyle birlikte 2D veya 3D grafikleri ekrana çizmek için kullanılan grafik uygulama geliştirme birimidir.Birçok programlama dilinde kullanılabilir.C dilinde kullanmak için kütüphaneyi indirip çeşitli yerlere taşımamız gerekiyor.
- ☺ IDE olarak ilk başta CodeBlocks'da başladıktan sonra orda bazı sorunlarla (verilen .txt uzantılı dosyayı okuyamama, windows.h kütüphanesinin stabil olmaması bazen eklemek zorunda kalıp bazen de eklemeyip ondan bağımsız da çalışabiliyordu) karşılaştıktan sonra Dev-C++ IDE'sini kullanmaya başladık.Burada her şey daha verimli çalışıyordu.

2. Yöntem:

- ☺ İlk başta yeniden_boyutlandir() adlı fonksiyonumuz ile koordinat düzlemimizin boyutunu 30x30 olarak ayarladık.Belirli OpenGL fonksiyonları ile koordinat düzlemimizi çizdirdik.
  (Bu fonksiyonların açıklamasına son sayfada yer verdik)
- ☺ Sonrasında dosyamızı okuma modunda açıp adet adındaki bir değişken yardımıyla dosyamızdaki verileri okuyup toplam kaç adet (x,y) ikilimiz olduğu bilgisini aldık.(adet bilgisini almak için void dosyadaki_nokta_adedi(int *adet) fonksiyonunu kullandık.)

```
while(!feof(fx))  {
        fscanf(fx,"%c %d %c %d %c
%c",&karakter,&x,&karakter,&y,&karakter,&karakter);
        *adet++;
    *adet-=1;}
```

- ☺ Daha sonra x,y değerlerimizi 2 boyutlu bir diziye atadık.Ve bu noktaları koordinat sistemimizde noktalalar halinde gösterdik.(void dosyadan_oku_ve_noktalari_ciz(int [][2],int) fonksiyonunu kullandık.)
- ☺ Sıra çember algoritmasına geldi.Bunun için(void yaricap_merkezx_merkezy_bul_yaricapi_merkezi_goster(float *,float *,float *,int [][2],int) fonksiyonunu kullandık.)Bizim kurduğumuz algoritmaya göre minimum çember olabilmesi için noktalardan en az 2sinin çember üzerinde bulunması gerektiği sonucuna vardık.İlk olarak birbirine en uzak 2 noktayı belirledik ve bu noktaların birbirine olan uzaklığının yarısını yarıçap olarak aldık.Bu 2 noktanın ortasını ise merkez olarak belirledik.

```
int karsilastir=adet*(adet-1)/2.0,bx=0;
int aradaki_uzaklik[karsilastir],xnoktalar[karsilastir][2],
```

```
ynoktalar[karsilastir][2];
for(int i=0; i<adet-1; i++)  {
for(int j=1; j<adet-i; j++) {
   aradaki_uzaklik[bx]=pow(dizi[i][0]-
dizi[j+i][0],2)+pow(dizi[i][1]-dizi[j+i][1],2);
         xnoktalar[bx][0]=dizi[i][0];
         xnoktalar[bx][1]=dizi[j+i][0];
         ynoktalar[bx][0]=dizi[i][1];
         ynoktalar[bx][1]=dizi[j+i][1];
         bx++;    }}
   int  enbuyuk=0,bizimx[2],bizimy[2];
   for(int i=0; i<bx; i++)  {
      if(aradaki_uzaklik[i]>enbuyuk)   {
         enbuyuk=aradaki_uzaklik[i];
         bizimx[0]=xnoktalar[i][0];
         bizimx[1]=xnoktalar[i][1];
         bizimy[0]=ynoktalar[i][0];
         bizimy[1]=ynoktalar[i][1];}}
 *merkezx=(bizimx[0]+bizimx[1])/2.0;
  *merkezy=(bizimy[0]+bizimy[1])/2.0;
```

- ☺ Yukarıdaki kod parçacığında tüm noktaların karşılaşma sayısını C(adet,2) karşılaştır adındaki değişkene atadık ve tüm uzaklıkları bulduk.Aynı zamanda karşılaşan x ve y leri de xnoktalar ve ynoktalar adlı 2boyutlu dizilere attık.Daha sonra bu uzaklıklardan en büyüğünü ve uzaklığın en büyük olduğu durumu karşılayan x ve y leri bulduk.Bu x y leri bizimx ve bizimy adlı diziye attık.
- ☺ Ancak bizim bulduğumuz yarıçap her zaman en küçük yarıçap olmuyordu.Çünkü bu algoritma çemberin üzerinde 2 nokta olduğu durumda doğru çalışıyordu.Bunu kontrol etmek için yaricapi_degis adında bir değişken tanımladık ve bulduğumuz merkezin tüm noktalara olan uzaklığını bulup bunların arasından en büyüğünü alıp yaricapi_degis adındaki değişkene atadık.Eğer yarıçapımız yaricapi_degis değerine eşitse bulduğumuz merkez ve yarıçap doğrudur ve çemberin üzerinde 2 nokta vardır.Eğer eşit değilse dışarda nokta kaldığını ve bu durumda çemberin üzerinde 2den fazla nokta olduğunu anlarız.Bu sorunu çözmek için de çevrel çembere başvururuz

```
float yaricapi_degis=0,r_ile_uzaklik;
int bulunan_x,bulunan_y;
   for(int i=0; i<adet; i++){
      r_ile_uzaklik=pow(dizi[i][0]-
*merkezx,2)+pow(dizi[i][1]-*merkezy,2);
      if(r_ile_uzaklik>=yaricapi_degis)
         yaricapi_degis=r_ile_uzaklik;    }
   float yaricap=sqrt(yaricapi_degis);
   if(*r==yaricap)
   {
     * r=yaricap;
     * merkezx=(bizimx[0]+bizimx[1])/2.0;
      *merkezy=(bizimy[0]+bizimy[1])/2.0;}
```

- ☺ Yukarıdaki kod parçasında bir önceki maddede bahsettiğimiz durumu gösterdik.Eğer yarıçapımız yarıcapi_degis değerine eşit değilse sorunu çevrel çemberden çözeceğiz ve bunun ikin 3.bir x y değerine ihtiyacımız var. Bu nedenle bulunanx ve bulunany adında 2 değişken tanımladık.
- ☺ Çemberin üzerinde olan 3.noktayı bulmak için önceden bulmuş olduğumuz çemberin üzerindeki diğer 2 noktadan yararlanacağız.Bu 2 noktanın orta noktasını buluyoruz ve bu orta noktaya en uzak olan nokta çemberin üzerindeki 3.noktamızdır.(Yani bulunanx ve bulunany.)

```
else   {
float
ortax=bizimx[0]+bizimx[1],ortay=bizimy[0]+bizimy[1],enb
uyukum=0,uzaklik;
for(int i=0; i<adet; i++)
{
if((dizi[i][0]!=bizimx[1]&&dizi[i][1]!=bizimy[1])||(dizi[i][0]!
=bizimx[0]&&dizi[i][1]!=bizimy[0]))
{
 uzaklik=pow(dizi[i][0]-(ortax/2.0),2)+pow(dizi[i][1]-
(ortay/2.0),2);
if(uzaklik>enbuyukum)
{
enbuyukum=uzaklik;
bulunan_x=dizi[i][0];
bulunan_y=dizi[i][1];
} }}
//Çevrel çemberin merkezini ve yarıçapını bulan
matematiksel formul.
float  akare=pow(bizimx[0],2)+pow(bizimy[0],2);
float bkare=pow(bizimx[1],2)+pow(bizimy[1],2);
float ckare=pow(bulunan_x,2)+pow(bulunan_y,2);
floatD=2*((bizimx[0]*(bizimy[1]-
bulunan_y))+(bizimx[1]*(bulunan_y-
bizimy[0]))+(bulunan_x*(bizimy[0]-bizimy[1])));
float merkezxim=(akare*(bizimy[1]-
bulunan_y)+bkare*(bulunan_y-
bizimy[0])+ckare*(bizimy[0]-bizimy[1]));
float merkezyim=(akare*(bulunan_x-
bizimx[1])+bkare*(bizimx[0]-bulunan_x)+ckare*(bizimx[1]-
bizimx[0]));
*merkezx=merkezxim/D;
 *merkezy=merkezyim/D;
 *r=sqrt(pow(bulunan_x-*merkezx,2)+pow(bulunan_y-
 *merkezy,2));}
```

- ☺ Yukarıdaki kod parçacığında ortax ve ortay(önceki 2 noktanın ortası) ye en uzak olan noktamızı belirledik ve bu noktayı bulunanx ve bulunany olarak atadık.
- ☺ Şimdi elimizde çemberin üzerinde olan 3 noktamız var.(3ten de fazla olabilir.Bunu çemberi çizdirdiğimizde otomatik olarak çemberin üzerinde göreceğiz.)

$$((A_y^2 + A_x^2)(B_y - C_y) + (B_y^2 + B_x^2)(C_y - A_y) + (C_y^2 + C_x^2)(A_y - B_y))/D,$$
$$((A_y^2 + A_x^2)(C_x - B_x) + (B_y^2 + B_x^2)(A_x - C_x) + (C_y^2 + C_x^2)(B_x - A_x))/D$$

$$D = 2(A_x(B_y - C_y) + B_x(C_y - A_y) + C_x(A_y - B_y)).$$

- ☺ Yukarıdaki formüle göre çevrel çemberin merkezini bulduran kod satırını yazdık.Daha sonra bu merkezle çemberin üzerindeki bir noktanın uzaklığını hesaplayıp çevrel çemberin yani minimum çevreleyen çemberin yarıçapını bulmuş olduk.

A:bizimx[0],bizimy[0];
B:bizimx[1],bizimy[1];
C:bulunanx,bulunany;

☺ Daha sonra bulduğumuz merkez ve yarıçapa göre minimum çevreleyen çemberi çizdirdik.(void cemberi_ciz(float,float,float) fonksiyonunu kullandık.)

☺ Sonrasında b-spline eğrisini araştırdık ve bulduğumuz verileri kendimize göre uyarlayarak noktanın en yakınınından veya üzerinden geçen eğriyi çizdirdik.Bunu yapmadan önce aynı noktaları farklı sırada verdiğimizde farklı bir eğri çizdirmemesi için noktalarımızı x değerlerine e göre sıraladık.

```
void diziyi_sirala(int dizi[][2],int adet){
float temp,temp1;
   for(int i = 0; i < adet ; i++){
      for(int j = 0; j < adet-i-1; j++) {
         if( dizi[j][0]> dizi[j+1][0]){
           temp = dizi[j][0];
            dizi[j][0]= dizi[j+1][0];
            dizi[j+1][0] = temp;
            temp1 = dizi[j][1];
            dizi[j][1]= dizi[j+1][1];
            dizi[j+1][1] = temp1;
         }}}}
```

Kübik rasyonel B-spline eğrisinin denklemini açık olarak yazalım;

$$P(t) = B_1R_{1,4}(t) + B_2R_{2,4}(t) + B_3R_{3,4}(t) + B_4R_{4,4}(t)$$

$$R_{i,k}(t) = \frac{h_i N_{i,k}(t)}{\sum_{i=1}^{n+1} h_i N_{i,k}(t)}$$ olduğundan öncelikle paydanın değerini hesaplayalım:

$$S = \sum_{i=1}^{n+1} h_i N_{i,k}(t) = h_1 N_{1,4} + h_2 N_{2,4} + h_3 N_{3,4} + h_4 N_{4,4} = (1-t)^3 + 6t(1-t)^2 + 3t^2 (1-t) +$$

$$S = 3t^3 - 6t^2 + 3t + 1$$

$$R_{1,4}(t) = \frac{h_1 N_{1,4}}{S} = \frac{(1-t)^3}{3t^3 - 6t^2 + 3t + 1}$$
$$R_{2,4}(t) = \frac{h_2 N_{2,4}}{S} = \frac{6t(1-t)^2}{3t^3 - 6t^2 + 3t + 1}$$
$$R_{3,4}(t) = \frac{h_3 N_{3,4}}{S} = \frac{3t^2(1-t)}{3t^3 - 6t^2 + 3t + 1}$$
$$R_{4,4}(t) = \frac{h_4 N_{4,4}}{S} = \frac{t^3}{3t^3 - 6t^2 + 3t + 1}$$

Bu eğrinin denklemi;

$$P(t) = B_1R_{1,4}(t) + B_2R_{2,4}(t) + B_3R_{3,4}(t) + B_4R_{4,4}(t)$$

☺ Yukarıda eğrimizi çizdirirken yararlanmış olduğumuz matematiksel ifadeyi belirttik.Bu formülü noktalarımıza uyarladık.

```
for(int i=0; i<adet; i+=3)
  {
     for( float a=0; a<=1; a+=0.001)
     {
        b =1-a;
 d=i+1;   m=i+2; g=i+3;
        if(i==adet-1)
        {
           d=i;   m=i; g=i;
        }
        if(i==adet-2)
        {
          m=i+1;  g=i+1;
        }
        if(i==adet-3)
        {
           g=i+2;
        }
        X
=(dizi[i][0]*b*b*b+dizi[d][0]*(6*a*b*b)+dizi[m][0]*(3*a*a*b)+dizi[g][0]*a*a*a)/(3*a*a*a-6*a*a+3*a+1);
        Y
=(dizi[i][1]*b*b*b+dizi[d][1]*(6*a*b*b)+dizi[m][1]*(3*a*a*b)+dizi[g][1]*a*a*a)/(3*a*a*a-6*a*a+3*a+1);
        glVertex2f(X,Y);
     }   }
  glEnd();
     glFlush();
```

☺ Yukarıda b-spline ı çizdirmek için yazdığımız kod var.Bu koda göre sıralanmış noktalarımıza (dosyada noktaların sırası farklı bile olsa aynı eğriyi çizdirmesi için ve daha düzgün bir şekil için)4 er 4er bakılıyor.Her seferinde i değerimiz +3 artıyor.Örneğin 12 noktadan oluşan bir dizimizi (0,3),(3,6),(6,9),(9,12) olmak üzere 4 kere dolaşır.

☺ B-spline eğrileri segmentlerden oluşur.Her segment için kontrol noktaları vardır.Eğrinin formunu belirlemede düğüm vektörleri de etkin rol alır.Eğri boyunca parametre değişimini gösteren düğüm vektörü eğrinin kaç polinom segmentinden oluşacağını belirler.Biz düğüm vektörünü 0-1 arasında aldık.Eğrimizi ona göre oluşturduk.(Araştırdığımız kaynaklarda düğüm vektörü hesaplamaları çok karışıktı.Eğriye bu vektör sayesinde istediğimiz şekli verebiliyoruz.Ancak biz tam olarak nasıl hesaplanacağını anlamadığımız için default olarak 0-1 aralığını seçtik.Eğrinin şeklini değiştirmek için düğüm vektörü hesaplanıp farklı aralıklar da alınabilir)

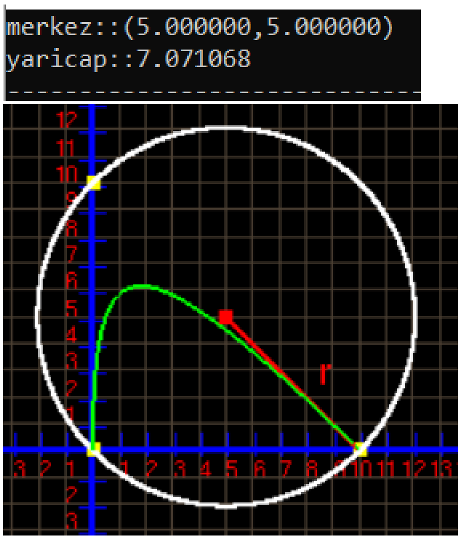
3. Kaba Kod/Sözde Kod:

1-Ana main fonksiyonuyla başla.
2-Pencere boyutunu ayarla ve pencereye isim ver.
3-Sekli_goruntule fonksiyonuna git ve sırasıyla koordinatı_birimlere_ayir,sayfayı_karelere_bol_koordinat_ciz_ok_ciz fonksiyonlarını uygula.
4-int adet=0;
5-Dosyadaki_nokta_adedi(&adet);
6.fopen(dosya,r)
7-while(!feof(dosya)){
oku(karakter,x,karakter,y,karakter,karakter) {0,3}, örneğin.
*adet++;}
fcloce(dosya);
*adet-=1;
8-int dizi[adet][2];
9-dosyadan_oku_ve_noktaları_ciz(dizi,adet);
10-fopen(dosya,r);
for(i=0;i<adet;i++){
oku(karakter,x,karakter,y,karakter,karakter)
Dizi[i][0]=x
Dizi[i][0]=y
çiz(x,y);}
fclose(dosya);
11-iny merkezx,merkezy,r;
12-Yaricap_merkezx_merkezy_bul_yaricapi_merkezi_goster(&r,&merkezx,&merkezy,dizi,adet);
13-Toplam ikili karşılaştırma sayısını bulup yeni oluşturduğun dizilere boyut olarak ver

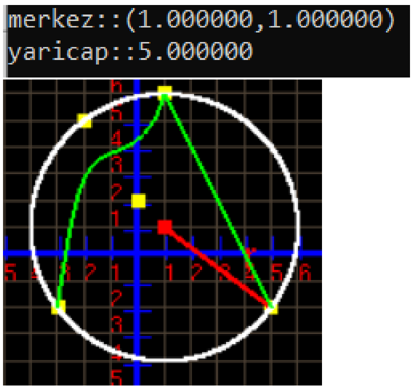
```
14-int
aradaki_uzaklik[karsilastir],xnoktalar[karsilastir][2],ynokta
lar[karsilastir][2];
15-for(i=0;i<adet;i++){
for(j=j=1; j<adet-i; j++){
2 nokta arasındaki uzaklığı bul.Bir dizide tut.
Her seferinde karşılaşan noktaları bir dizide tut.
}}
16-Enbüyük diye değişken belirle ve en uzak iki noktayı
temsil eden dizileri tanımla.
17-int enbuyuk=0,bizimx[2],bizimy[2];
 For(i=0;i<karşılaşma sayısı;i++){
 if(aradaki uzaklık[i]>Enbuyuk){
     enbuyuk=aradaki uzaklık;
          bizimx[0]=xnoktalar[i][0];
            bizimx[1]=xnoktalar[i][1];
            bizimy[0]=ynoktalar[i][0];
            bizimy[1]=ynoktalar[i][1];}
18- *merkezx=(bizimx[0]+bizimx[1])/2.0;
    *merkezy=(bizimy[0]+bizimy[1])/2.0;
    *r=karekökü((*merkezx-bizimx[0])karesi+(*merkezy-
bizimy[0]karesi);
19-float yaricapıdegis=0,r_ile_uzaklık;
20- int bulunan_x,bulunan_y;
21-for(i=0; i<adet; i++){
 r_ile=uzaklık=i.noktamızın merkeze olan uzaklığı
        if(r_ile_uzaklik>=yaricapi_degis)
            yaricapi_degis=r_ile_uzaklik;
   }
int yaricap=yarıcapı_degisin karekökünü al.
22-if(*r==yaricap)->Eğer değişen yarıçap ilk yarıçapa eşitse
çemberin üzerinde 2 nokta vardır ve bulduğumuz merkez
ve yarıçap çembere aittir.
23-Else-->Eğer değişen yarıçap ilk yarıçaptan büyükse
çevrel çember devreye girsin.{
24-Bulduğunuz merkeze en uzak olan bir başka nokta
daha bul.
25- uzaklik=karesi(dizi[i][0]-(merkezx),2)+karesi(dizi[i][1]-
(merkezy),2);
for(i=0;i<adet;i++){
                if(uzaklik>enbuyukum)
                {
                    enbuyukum=uzaklik;
                    bulunan_x=dizi[i][0];
                    bulunan_y=dizi[i][1];
                }
}
26-Çevrel çemberin merkezini ve yarıçapını bulan
mateamtiksel formüle başvur.
Raporda yer verilmiştir.
*merkezx,*merkezy,*r yeni değerleri bul.
27-çiz(merkez);
28-çiz(yarıçap);
29-Çemberi_ciz(merkezx,merkezy,r);
30-for(j=-1.0;j<1.0;j++){
çiz(merkezx+cos(j*pi sayısı)*r,merkezy+sin(j*pisayısı)*r) ;
31-diziyi_sırala(dizi,adet);
32-int temp,temp1;
33-for(i=0;i<adet;i++) {
 for(j=0;j<adet-i-1;j++){
if(dizi[j][0]>dizi[j+1][0]){
 dizilerin yerlerini değiştir.}
34-float X,Y;
35-çizimi başlat.
36-int d,m,g;
37-for(i=0;i<adet+i++){
 for(a=0;a<=1;a+=0.001){
X=b-spline formulü uygula;
Y=b-spline formulü uygula;
çiz(X,Y)}}
  Yaz(Konsal ekranına merkezin x ve y'sini yazdır.)
                    Yaz (Yarıcapı da altına yazdır.)
```

4. Deneysel Sonuçlar ve Sonuç:

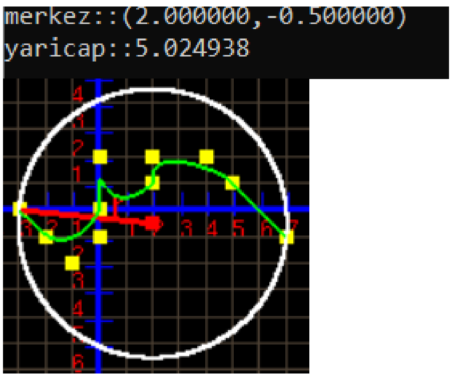
Bize verilen proje dökümanındaki ve ayriyyeten internetten bulduğumuz noktalarda projemizin merkez ve yarıçapı doğru bulduğunu gördük.B-spline eğrimiz ise çoğu noktadan geçmekle birlikte bazı noktaların yakınından geçiyor.



{0,0},{10,0},{0,10},



{1,6},{5,-2},{0,2},{-2,5},{-3,-2},



{-1,-2},{-3,0},{2,1},{7,-1},{0,0},{4,2},{2,2},{5,1},{0,2},{0,-1},{-2,-1},

5. Big O Analizi:

Yazılan bir algoritmanın performansını ölçebilmemiz için kullanacağımız en önemli araçlardan biri Big-O notasyonudur. Hesaplamamızı yaparken for döngülerine bakmayı tercih edeceğiz.Çünkü int x=1; gibi atamalar 1 işlem e eşit ve bu gibi 1 işlemler n^2 ler n! lerin yanında önemsenmeyecek sabitlerdir.

- ✓ for(int i=0; i<adet; i+=3)
  { for(float a=0; a<=1; a+=0.001)
  {sabit sayıda işlemler}}
- ✓ for(int i =1; i<30; i++)
  {sabit sayıda işlem}
- ✓ for(int i =0; i<10; i++)
  {sabit sayıda işlem}
- ✓ for(int i =0; i<42; i+=2)
  {sabit sayıda işlem}
- ✓ for(int i=0; i<adet; i++)
  {sabit sayıda işlem}
- ✓ for(int i=0; i<adet-1; i++)
  {for(int j=1; j<adet-i; j++)
  { sabit sayıda işlem}}
- ✓ for(int i = 0; i < adet ; i++)
  {for(int j = 0; j < adet-i-1; j++)
  {sabit sayıda işlem}}
- ✓ for(float j=-1.0 ; j<=1.0 ; j+=0.0001)
  {sabit sayıda işlem}

Yukarıda projemizde kullanmış olduğumuz döngüleri gösterdik.Bizim big-o notasyonumuzu belirleyecek olan karmaşıklığı en büyük çıkan for olacaktır.Ana karmaşıklık n(adet) e bağlı olacağından dolayı n(adet) e bağlı olan for ları hesaplayacağız.Bu durumda yukarıda belirttiğimiz for döngülerinden 6. Ve 7.yi ele alacağız.Önce 6.yı hesaplayalım…

- i=0 iken j=adet-0 a kadar dönecek.Yani n iş yapar.
- İ=1 iken j=adet-1 e kadar döner.Yani n-1 iş yapar.
- İ=2 iken j=adet-2 e kadar döner.Yani n-2 iş yapar.
- İ=3 iken j=adet-3 e kadar döner.Yani n-3 iş yapar.
- İ=4 iken j=adet-4 e kadar döner.Yani n-4 iş yapar.
- İ=adet-2 iken j=adet-(adet-2) e kadar döner.Yani 2 iş yapar.

2+3+4+5+6+…(n-1)+(n) kadar iş yapılmış olur.Bunun karmaşıklığını da ardışık terimler toplamından (n+1)*(n)/2 gelir.Tabii bunun bir katsayısı da vardır.(a*n^2+b*n+c)/2 dir sonucumuz.BigO gösterimi ise O(n^2)dir.

- i=0 iken j=adet-0-1 a kadar döner.Yani n-1 iş yapar.
- İ=1 iken j=adet-1-1 e kadar döner.Yani n-2 iş yapar.
- İ=2 iken j=adet-2-1 e kadar döner.Yani n-3 iş yapar.
- İ=3 iken j=adet-3-1 e kadar döner.Yani n-4 iş yapar.
- İ=4 iken j=adet-4-1 e kadar döner.Yani n-5 iş yapar.
- İ=adet-1 iken j=adet-(adet-1)-1 e kadar döner.Yani 0 iş yapar.

0+1+2+3+4+5+….+(n-2)+(n-1) kadar iş yapılmış olur. Bunun karmaşıklığını da ardışık terimler toplamından (n-1)*(n)/2 gelir.Tabii bunun bir katsayısı da vardır.(d*n^2-e*n+f)/2 dir sonucumuz.BigO gösterimi ise O(n^2)dir.

Sonuç olarak elimizde diğer döngülerde yapılan C kadar +karmaşıklığı büyük olan 2 döngüde yapılan (a*n^2+b*n+c)/2 ve (d*n^2-e*n+f)/2 olmak üzere 3 bilgi var.
Biz bunların önündeki katsayılara bakmadan en büyük dereceli terimi alıyoruz.Burada en büyük dereceli terimimiz n^2 dir.Dolayısıyla bizim big-o muz O(n^2)dir.

6. Kullandığımız Opengl Çizim Fonksiyonları:

- glutInit();
- glutInitDisplayMode();
- glutInitWindowSize();
- glutInitWindowPosition();
- glutCreateWindow();
- glutDisplayFunc();
- glutReshapeFunc();
- glutMainLoop();
- glClear();
- glColor3f();
- glLineWidth();
- glBegin();
- glEnd();
- glFlush();
- glutSwapBuffers();
- glMatrixMode();
- glOrtho();
- glRasterPos2f();
- glutBitmapCharacter();
- glVertex2f()

7. Kaynakça:

- https://tr.wikipedia.org/wiki/%C3%87evrel_%C3%A7ember#%C3%87evrel_%C3%A7ember_merkezinin_koordinatlar%C4%B1
- https://www.youtube.com/watch?v=wG_VaSr6a6c
- http://bilgisayarkavramlari.sadievrenseker.com/2009/08/10/splines-seritler/
- http://earsiv.atauni.edu.tr/xmlui/bitstream/handle/123456789/1347/10067779.pdf?sequence=1
- http://www.electropazar.com.tr/bilgibankasi/opengl.html
- https://en.wikipedia.org/wiki/Smallest-circle_problem
- https://en.wikipedia.org/wiki/B-spline
- https://www.geeksforgeeks.org/