PROGRAMLAMA LABORATUVARI 1

1. PROJE

Yunus Emre GÜL
Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi
yunemregul@gmail.com

Özet

Bu doküman Programlama Laboratuvarı 1 dersi 1. Projesi için çözümümü açıklamaya yönelik oluşturulmuştur. Dökümanda projenin tanımı, çözüme yönelik yapılan araştırmalar, kullanılan yöntemler, proje hazırlanırken kullanılan geliştirme ortamı ve kod bilgisi gibi programın oluşumunu açıklayan başlıklara yer verilmiştir. Doküman sonunda projemi hazırlarken kullandığım kaynaklar ve proje derlenirken dikkat edilmesi gereken hususlar bulunmaktadır.

1. Proje Tanımı

Projede bizden istenen, C dili kullanarak belirli bir klasörün içinde bulunan ".nkt" uzantılı tüm dosyalardan ayrı ayrı 3 boyutlu nokta verilerini okuyan, üzerlerinde belirli işlemleri yapıp sonuçları çıktı olarak başka bir dosyaya kaydedecek programı yazmaktır. Program dosyalardaki bilgileri işleyip ardından belirli isterleri kullanıcının isteği üzerine gerçekleştirecektir.

Dosyalar sabit bir formatta verilmiştir. Dosyalar 5 satırlık bir başlık bilgisi ile başlayıp ardından nokta verileri ile devam etmektedir. Başlık bilgisinin sırası ve içeriği sabit formatı takip etmelidir. İlk satırda "#" işareti ile başlayan bir yorum-açıklama satırı bulunmaktadır. Ardından dosyanın "VERSION" bilgisi ve devamı aşağıdaki Şekil 1 de verilmiştir:

- # Noktalar dosya formatı
- VERSION 1
- ALANLAR x y z r g b
- NOKTALAR 213
- DATA ascii

Başlık Bilgileri

Şekil 1. Başlık Formatı

Programın çalıştığı klasördeki herhangi bir ".nkt" uzantılı dosyanın başlık bilgisi yukarıda gösterilen formata uygun değilse dosyadan okuma yapılmaması ve ilgili hata mesajı verilmesi gerekir. Formatta "VERSION" bölmesi sabit olarak 1, "DATA" bölmesi "ascii" ve "binary" arasında değişebilmekte, "ALANLAR" bölmesi ise "x y z" ve "x y z r g b" arasında değişim gösterebilmektedir.

Dosyanın "DATA" bölmesi "ascii" ise dosyadaki noktaların koordinat bilgileri ASCII formatında, "binary" ise BINARY formatında saklandığı belirtilmektedir. Bu iki farklı saklama tipi için de farklı okuma yöntemleri kullanılması gerekir.

Dosyanın "ALANLAR" bölmesi "x y z" şeklinde belirtilmiş ise noktaların 3 boyutta bilgilerinin verildiği kabul edilir. Eğer "x y z r g b" şeklinde belirtilmiş ise noktaların 3 boyutta bilgilerinin yanında renk bilgisi de kırmızı yeşil mavi kodları ile birlikte verildiği kabul edilir. Bu renk bilgisini de herhangi bir noktayı yazdırırken vermemiz gerekmektedir.

Dosyadan okuma yapıldıktan sonra okunan nokta sayısı başlık kısmındaki "NOKTALAR" sayı bilgisi ile aynı değilse uygun hata mesajı yazdırılmalı ve dosyadan okuma yapılmamalıdır.

Dosyadaki noktaların "x y z" koordinat bilgileri *float* formatında, "r g b" renk bilgileri ise *int* formatında saklanmaktadır.

Dosyadaki noktaların bilgileri kaydedildikten sonra bir menü aracılığı ile kullanıcının seçimi üzerine

aşağıdaki isterler tüm dosyalarda gerçekleştirilip sonucu hem konsol ekranında gösterilmeli, hem de bir çıktı dosyasına kaydedilmelidir.

1. Kullanıcı 1 tuşuna bastığında tüm dosyaların uyumlu olduğunu ya da hangi dosyalar hatalı ise hata bilgileri,

2. Kullanıcı 2 tuşuna bastığında tüm noktalar arasında birbirine en yakın ve en uzak noktaların bilgileri ve nokta numaraları,

3. Kullanıcı 3 tuşuna bastığında tüm noktaları içine alacak en küçük küpün köşe nokta bilgileri,

4. Kullanıcı 4 tuşuna bastığında kullanıcıdan XYZ koordinat bilgileri ve R yarıçap bilgisi alınmış bir kürenin içinde kalan tüm noktalar,

5. Kullanıcı 5 tuşuna bastığında her bir noktanın birbirine olan uzaklıklarının ortalaması yazdırılmalı ve çıktı dosyasına kaydedilmelidir.

2. Araştırmalar ve Yöntem

Projede tüm dosyaları okuyabilmek için öncelikli olarak programın çalıştığı klasördeki tüm dosyaların bulunabilmesi gerekiyor. Bu sorunu çözmek için araştırmalar yapıp projeye en uygun olduğunu düşündüğüm yöntemi kullandım. Kullandığım yöntem klasördeki tüm dosyaların ismine ulaşmamı sağladı. İsimlerini bildiğim dosyaları da rahatlıkla okuyabildim.

Programın çalıştığı klasördeki tüm dosyalara ulaşınca içindeki verileri değerlendirmenin en iyi yolunun dosyayı satır satır okutmak olduğuna karar verdim böylece dosyayı parça parça değerlendirebilmiş olacaktım ve bu da hata tespitinde kolaylık sağlayacaktı ancak bu yöntem BINARY olarak verilmiş dosyalarda noktaları okurken geçersiz olacaktı. Yine de ASCII dosyalar için satır satır okuyarak başlığı ve nokta bilgilerini değerlendirdim. BINARY dosyalarda ise yine başlığı ASCII deki gibi satır satır değerlendirip noktaların verildiği BINARY kısım için binary okuma yaptım. Böylece iki sorun da çözülmüş oldu.

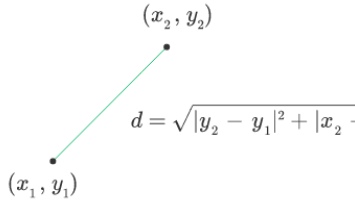
Bir başka problem ise okunan bu nokta ve başlık bilgilerinin nasıl saklanacağını kararlaştırmaktı. Burada her dosyadaki bilgileri farklı farklı değişkenlerle saklayabileceğimi düşündüm ama bu yöntem hem çok düzensiz hem de kullanışsız olacaktı. Bunun yerine tüm dosyaları ve başlık bilgilerini, dosyadaki noktaları birlikte tutabilecek bir *struct* yani bir yapı tanımladım, bu yapıyı da her yeni dosya okunurken çoğalttım. Böylece tanımladığım yapı gerçek dosyaları birebir temsil edecek şekilde oldu. Yapı aşağıdaki şekilde gösterilebilir.

```
struct Dosya
{
char Ad[128];
struct DosyaBaslik Baslik;
float *Noktalar;
int OkunanNokta;
}
```

Bu sayede dosya bilgilerine erişim de kolaylaşmış oldu.

Dosya bilgilerini ve dosyalardaki noktaları saklarken denk geldiğim bir diğer problem ise dosya sayısının ve okunacak nokta sayısının belirli olmamasıydı. Okunacak nokta sayısı kısmen başlıkta belli olsa da başlık 10.000 nokta okuyacağını söyleyip aslında dosyada 5 nokta olabilmesi gibi durumlar söz konusu idi. Böyle bir durumda 9.995 boyutluk bellek boşuna ayrılmış olacaktı. Bunu çözmek için de dinamik bellek ataması kullandım. Her yeni dosyayı okurken dosyaları kaydettiğim yapıyı çoğaltarak uygun bir saklama yöntemi elde ettim. Yine dosyalardaki noktaları okurken de dosya yapısının içindeki noktaların boyutunu dinamik bellek ataması ile çoğaltarak yeni nokta okunmasını sağladım. Böylece boyutu önceden tanımlı sabit bir dizi kullanmak yerine boyutu okunan her noktayla birlikte değişen bir dizim olmuş oldu. Bu da programa hem bellek tasarrufu sağladı hem de değişen koşullara uyumluluğunu arttırdı.

Bütün okuma işlemlerini gerçekleştirdikten sonra geriye matematiksel hesaplamalar kaldı. İlk olarak noktalar arasındaki mesafeyi bulma işlemi neredeyse her ister tarafından kullanıldığından bunu bir fonksiyon olarak tanımlamanın daha doğru olacağına karar verdim. İki boyutta noktaların mesafesini bulan temel formülü üç boyuta uyarlamak yeterli oldu.



Şekil 2. 2 Boyutta Noktalar Arası Mesafe

İsterler arasında en çok vakit harcadığım tüm noktaları içine alacak olan küpü bulmam gereken

ister idi. Bunun üzerine ilk düşündüğüm şey dosyadaki bütün noktaların eksenlerdeki maksimum ve minimum sınırlarını bularak bir küp oluşturabileceğim oldu. Başta bu yöntemi uygulamanın çok kompleks olacağını sanıp başka yöntemlere göz attım, en çok vakti de başka yöntemlere bakarak harcamış oldum. Sonuç olarak yine ilk düşündüğüm yöntemi uygulamak zorunda kaldım. Yöntemin detaylı anlatımı bölüm 4.2. de bulunabilir.

Bir başka vakit harcadığım kısım ise en yakın ve en uzak noktaları bulan algoritmanın yavaşlığı idi. Başta en kısa ve en uzak noktaları bulmayı ayrı iki fonksiyon olarak tanımlamıştım. Bu iki fonksiyonu tek bir fonksiyonda birleştirmek işlem yükünü iki kat azalttı. Daha sonrasında algoritmamdaki gereksiz işlemleri azaltarak 5-6 dakika süren hesaplamayı 10 saniyelere kadar indirdim. Derleme yaparken kullandığım optimizasyon konfigürasyonu sayesinde de başta çok yavaş olan bu hesaplama bir hayli hızlandı.

Yine karşılaştığım bir başka sorun ise örnek olarak verilen *veriler2.nkt* dosyasındaki noktaların ortalamasını hesaplamaktı. Ortalama bulma algoritmama göre önce tüm noktaların birbirine mesafesi bir yere toplanıyor ardından da kaç mesafe var ise o sayıya bölünüyordu. Ancak bu dosyada da yaşadığım sorun dosyada yaklaşık 85.000 nokta olduğundan tüm noktaların birbirine mesafesi toplanırken *float* veri tipi bu toplamı taşıyamıyordu, boyutu yetmiyordu. Bu da noktaların ortalama değerinin -5 gibi manıksız bir değer yapıyordu. Bunu da çözmek için bir süre araştırma yaptım ve en uygun yöntemin *iterative mean* olarak bilinen bir yöntem olduğunu öğrendim. Bu yönteme göre tüm mesafeler bir yere toplanmaksızın her yeni mesafe ortalamaya eklenişinde mesafenin kendisi değil ortalamaya katkısı toplanıyor, bu da *float* veri tipinin taşmasını önlüyor. Algoritmayla ilgili detay bölüm 4.2. de bulunmaktadır.

Denk geldiğim diğer ufak bir problem ise kullanıcıdan seçim alırken *stdin* bufferinin temizlenmemesiydi, bu problem *fflush* fonksiyonu ile çözülebilmekte ancak Linux üzerindeki denemelerimde *fflush* çalışmadığı için araştırma yaparak aynı görevi gören alternatif bir fonksiyon oluşturdum. (clean_stdin)

3. Geliştirme Ortamı

Projemi Linux sistemde, Visual Studio Code OSS üzerinde geliştirip GCC kullanarak derledim. Projemin gelişimini ve versiyonlarını takip edebilmek için de Git versiyon kontrol sistemi kullandım.

4. Kod Bilgisi

4.1. Akış Şeması

Kısım ektedir. [1](akis-diyagrami.jpg)

4.2. Algoritma

Bu kısımda projenin genel algoritmasına açıklık getireceğim.

Program çalıştığında, olduğu klasördeki tüm dosyaları buluyor. Bulduğu dosyaların isimlerinin son kısmına bakarak eğer ".nkt" iseler bunları değerlendirmeye alıyor. Değerlendirmeye alınan her dosya satır satır okunmaya başlanıp ilk karakteri "#" olan yani yorum satırı olan satırları geçerek başlık bilgisinin başladığı ilk satırı buluyor. Başlık bilgisinin bulunduğu ilk satırın ve arkasındaki satırların programda tanımlı olan başlık sırasıyla verilip verilmediği, başlık bilgisinin uygun olup olmadığı tespit ediliyor. Eğer herhangi bir hata var ise dosya okunmayı bırakıp hatalı dosya için hata kaydediliyor. Hata yok ise program başlık bitene kadar dosyayı okuyor ve başlıktaki karakter sayısını toplayarak kaydediyor. Bu bilgiyi ilerde binary datasını okurken kullanacağız.

Başlık okunması bittiğinde dosya hakkında bilmemiz gerekenleri biliyor oluyoruz. Eğer dosyanın data tipi ASCII şeklinde verildi ise başlık bittikten sonra da satır satır okumaya devam ediyoruz. Her satırı boşluklara göre bölüp parçaların sayı olup olmadığını kontrol ediyoruz, tümünün sayı olması gerektiğinden sayı değiller ise ilgili hata mesajını daha sonra yazdırılmak için kaydediliyor. Sayı olmamaları dışında bir diğer olasılıkta hiç bulunmamaları, bu durum için de ilgili hata mesajı kaydediliyor ve dosya okunması bırakılıyor. Hiçbir sorun yoksa ve satırdaki parçalar düzgünse her parçayı floata çevirip ilgili dosyanın ilgili noktasına kaydediyoruz.

Eğer dosyanın data tipi BINARY şeklinde verildi ise, dosyanın başından başlıktaki karakter sayısınca uzaklaşıp tam oradan yani başlığın bittiği yerden *float* verisi okuyup ilgili dosyanın ilgili noktasına kayıt yapıyoruz. Dosya "x y z" şeklinde verildi ise arka arkaya 3 tane *float,* dosya "x y z r g b" şeklinde verildi ise arka arkaya 3 tane *float* 3 tane de *int* verisi okuyoruz.

Noktaları saklarken "x y z r g b" değişkenlerini ayrı ayrı saklamak bu değerlere ulaşımı döngülerde ve daha birçok yerde zorlaştıracaktı. Bunun yerine tüm

nokta verilerini *float* dizilerde saklamaya karar verdim. Buna göre;

Noktalar[0] = x;

Noktalar[1] = y;

Noktalar[2] = z;

Noktalar[3] = r;

...

şeklinde sakladım. Eğer dosya "x y z" şeklinde verildi ise dosyanın noktalar dizisinde her 3 *float* 1 noktayı, eğer dosya "x y z r g b" şeklinde verildiyse her 6 *float* 1 noktayı temsil ediyor. Böylece tek döngüyle bir dosyadaki tüm noktaların tüm verilerine ulaşılabiliyor.

İsterler kısmına gelecek olursak, en yakın en uzak noktaları bulma algoritması tüm noktaları karşılaştırıp aralarındaki mesafe şimdiye kadar bulunan en küçükten daha küçükse, yeni en küçük mesafe ve en yakın noktalar tanımlanmış oluyor. En uzak için de benzer şekilde.

Tüm noktaları içine alacak küp algoritmasında bütün noktalar bir döngü ile gezilip eksenlerde en küçük *x*, en büyük *x*, en küçük *y*, en büyük *y*, en küçük *z*, en büyük *z* bilgileri bulunuyor. Sadece bu bilgilerle 3 boyutlu bir şekil oluşturacak olursak dikdörtgenler prizması oluşacağından bu prizmayı küpe çevirmemiz gerekiyor. Bunu yapmak için de bu noktalar içinde aralarında en büyük fark olanı bulup aralarındaki fark daha küçük olanları bulduğumuz en büyük farka denk hale getirecek şekilde aralarını eşit şekilde açıyoruz. Örneğin birbirine en uzak olanlar en küçük *z* ile en büyük *z* bilgisi ise, *x* lerin arasındaki farkı denkleştirmek için aşağıdaki işlem yapılır:

en küçük x = en küçük x – (zfark-xfark)/2

en büyük x = en büyük x + (zfark-xfark)/2

Buradaki *zfark* en büyük *z* ile en küçük *z* arasındaki farkı temsil etmektedir, x*fark* ta benzer şekilde.

Merkezinin 3 boyutlu koordinatları ve yarıçap bilgisi kullanıcıdan alınan kürenin içinde kalan tüm noktaları bulmak için tüm noktaların kullanıcıdan alınan *xyz* noktasına uzaklığının yarıçaptan ufak olup olmadığına bakmamız yeterlidir. Yarıçaptan ufaksa kürenin içinde kalıyor anlamına gelir.

Her bir noktanın birbirine olan uzaklıklarının ortalamasını bulan ortalama bölüm 2. de bahsedildiği gibi *iterative mean* yani "iteratif ortalama" olarak bilinen matematiksel algoritmadır. Elde edilişi ise şu şekildedir:

$$\langle x \rangle_t = \frac{1}{t}\sum_{i=1}^{t} x_i$$

$$= \frac{1}{t}\sum_{i=1}^{t-1} x_i + \frac{1}{t}\,x_t$$

$$= \frac{t-1}{t}\;\frac{1}{t-1}\sum_{i=1}^{t-1} x_i +$$

$$= \frac{t-1}{t}\langle x \rangle_{t-1} + \frac{1}{t}\,x_t .$$

Şekil 3. "*Iterative Mean"* hesabının elde edilişi

Burada $\langle x \rangle_t$ olarak tanımlanan, *t* adet değerin ortalaması anlamına gelmektedir. Bu algoritma sayesinde tüm değerleri bir yere toplamak yerine değerlerin ortalamaya katkısı toplanmaktadır. Böylece toplamanın yapıldığı veri tipinin taşması önlenmektedir.

4.3. İstatistik

Program kodu boşluksuz ve yorumsuz yaklaşık 720 satır tek dosyadan oluşmaktadır. Kod düzenini sağlamak için yaklaşık 145 boş satır kullanılmıştır. Okuyucuya izlenim oluşturması için yaklaşık 90 yorum satırına yer verilmiştir.

Kullandığım kütüphaneler ve ne için kullandığım kabaca aşağıdaki gibidir:

<stdio.h>

Çıktı ve girdi almak için

<stdlib.h>

malloc, realloc, free gibi fonksiyonlar için

<dirent.h>

Programın çalıştığı klasördeki tüm dosyaları bulmak için

<string.h>

Klasörün uzantısını bulmak gibi *"str..."* fonksiyonları için

<math.h>

pow, sqrt gibi fonksiyonlar için

<ctype.h>

isdigit, toupper, iscntrl gibi fonksiyonlar için

4.4. Programın Derlenmesi

Programın kaynak kodu tek dosyadan oluşmaktadır. Bu dosyayı CodeBlocks ya da GCC kullanarak derleyebilirsiniz. Derlerken dikkat edilmesi gereken mesele ise eğer CodeBlocks kullanıyorsanız

Release modunda derlediğinize emin olun. Bu, programa ekstra optimizasyon ve hız katacaktır. Ayrıca CodeBlocksun bazı versiyonları kodun C99 standardında yazıldığına dair bir hata verebilir. Bunu düzeltmek için projenin üzerine sağ tıklayıp *"Build Options"* sekmesine girip *"Other compiler options" bölümüne*

-std=c99

eklemelisiniz.

Kaynakça

1. Bir klasördeki tüm dosyaları bulmak, https://www.geeksforgeeks.org/c-program-list-files-sub-directories-directory/

2. Heiko Hoffman, "Iterative Mean Algorithm", http://www.heikohoffmann.de/htmlthesis/node134.html

3. *fflush* fonksiyonuna alternatif, https://stackoverflow.com/a/17319153/8993088