INF103 ALGORİTMA VE İLERİ PROGRAMLAMA
PROJE RAPORU

Mayıs 30, 2021

Bengü Yurdakul	**19401831**
Çisem Kaplan	**19401836**
Doruk Bulut	**20401906**

İçindekiler

GİRİŞ

Günümüzde telefonlar ve kameralar insanların vazgeçilmezi haline gelmiştir. Bu iki araç sayesinde anlarımızı ölümsüzleştirebiliyor, çeşitli sosyal medya platformlarında hayatımız ve kendimizle ilgili detayları paylaşabiliyoruz. Sadece eğlence maksatlı değil habercilikte, adli vakalarda ve eğitimde sıkı sıkıya fotoğraflara ihtiyaç hissediyoruz.

Fotoğraf makinesinin icadından 2012 yılına kadar 3.5 trilyon fotoğrafın çekildiğini göz önünde bulundurursak bu kadar fotoğrafın hatasız ve güzel çekilmiş olması imkansızdır. Bu tür durumlarda fotoğraflarımızla ilgili düzenlemeler yapmak, rengini ve şeklini ayarlayarak onları istediğimiz şekle sokmak için çeşitli uygulamalara ihtiyaç duyarız.

İşte yapılan bu proje bu özellikleri içeren, fotoğraflarla ilgili birçok değişikliğe izin veren bir programdır. Peki bu program hangi değişikliklere imkân verir? İlk uygulama fotoğraftaki piksellerin %20'sinin piksel değerini rastegele değiştirir; ikinci uygulama siyah beyaz fotoğraflarda siyah pikseli beyaz, beyaz pikseli de siyah yapar; üçüncü uygulama belli piksel değerinin altındaki değerleri siyah üzerinde kalanları da beyaz yapar; dördüncü uygulama resme ayna efekti uygular. Son uygulama ise fotoğrafı saat yönünde doksan derece döndürür. Tüm bu uygulamalar birden fazla kez çalıştırılabilir.

Gerçek hayatta kullanılan uygulamalardan biriyle bu proje arasındaki farklardan biri siyah pikseli beyaz beyaz pikseli siyah yapma uygulamasıyla ilgili. Gerçek uygulamada sadece fotoğraftaki insan figürü değişirken bu projede fotoğrafın tümüne bu uygulanabilir. Bununla birlikte bulunan diğer çözümler en etkili biçimde yapıldığı için gerçek hayattaki uygulamalarla büyük benzerlik göstermektedir.

PROJE MİMARİSİ

Projede kullandığımız teknolojilerin temeli, fotoğrafların, makine tarafından nasıl algılandığı ile ilgilidir. Temel olarak fotoğraf dosyaları, renk değerlerinin sayılar ile ifade edildikleri dosyalardır.Her bir renk değerine pixel ismi verilir. Bir A4 kağıt alın ve eşit ölçekli, 2x2 boyutunda bir tablo çizin. Bu noktada, "2x2" ile ifade edilen sayı resmimizin en*boy bilgisidir. Tablonun her bir kutusuna "pixel" adı verilir ve içlerine renkleri ifade eden değerler yazılıdır. Makine ilgili sayılarla renk değerlerini eşleştirildiğinde ortaya resmimiz çıkar. (Dikkat etmek gereken bir nokta ise bu tablonun makinede depolanma şeklidir. A4 üzerine çizdiğiniz 2x2 boyutunda bir resmi, makine en boy ayrımı yapmadan, sıralı dört kutucuk şeklinde tutar; satır atlama karakterleri sayesinde kutucukları uygun pozisyonlara yerleştirir.) Pixel olarak ifade edilen her bir kutucuk rastgele sayılar almaz. Projemiz kapsamında, siyah-beyaz bir resim için 0-255 aralığından (tam sayı) değer alınır. 0 sayısı siyahı ifade ederken, 127 ve 255 sayıları sırasıyla gri ve beyaz renklerini ifade eder. Haliyle, siyahtan beyaza giden bir renk yelpazesi oluşturulur.

Fotoğraf dosyalarının ne olduğunu tanımladıktan sonra, yazdığımız programın bu dosyalara erişmesi, içeriğini okuması ve istediğimiz yönde değişiklikleri uygulaması gerekmekteydi. İlk olarak dosyaları C dilinde ifade etmek için, "PGMInfo" isminde bir veri yapısı tanımladık. Bu yapıda, siyah-beyaz bir fotoğraf dosyasının her bir özelliğini depolayan değişkenler kullandık (yorum, max-pixel değeri, en-boy değerleri ve pixel değerleri).

Veri yapısını tanımladıktan sonra dosyalara, sonrasında üzerinde çalışmak amacıyla erişip okuma-yazma işlemlerini yapmamız gerekiyordu. Bunun için C dilinin dosya işlemleri özelliğinden yararlanıp pgm_write ve pgm_read olmak üzere iki ayrı fonksiyon

kodladık. PGM Read fonksiyonu istenilen dosyaya erişip içindeki bilgileri PGMInfo veri yapısına depolanmaktadır (cf. pgm.c->pgm_read()). pgm_write ise üzerinde istenilen işlem gerçekleştirilen fotoğrafın bilgilerini yeni bir dosyaya yazmaktadır.

Dosyalara ve içeriklerine eriştikten sonra üzerlerinde uygulanacak işlemler için çeşitli fonksiyonlar geliştirdik.

Efekt uygulamasının temel amacı, kullanıcı tarafından seçilen bir fotoğrafa yine kullanıcı tarafından seçilen efekti uygulamasıdır.Bunun için pixellerin tutulduğu dizileri baştan sona gezip gerekli değişiklikleri yapacak döngüleri kullandık. Döngülerinin gezme aralıkları ise fotoğrafın en ve boy bilgilerine bağlıdır.

Projemizin ikinci uygulaması olan median filtrelemenin temel amacı bir fotoğraftan seçilen bir öğeyi temizlemektir. Çalışma prensibi aynı açıdan çekilmiş çokça fotoğraf ve bir sıralama algoritmasına dayanır. Bu aynı açıdan çekilmiş eşantiyon fotoğrafların her bir pixeli kendi içinde sıralanır. Sıralanan dizinin ortanca değeri, en çok tekrar eden değer ve dolayısıyla olasılıksal olarak ögesiz pixel olacaktır (çünkü öğeli pixel resim örüntüsünü bozacaktır). Filtrelenmiş fotoğrafın ilgili pixeli için bu ortanca değeri seçersek, filtrelenmek istenen ögenin bulunmadığı pixel değerini elde ederiz.

Programımıza çok sayıda eşantiyon fotoğrafın her bir pixelini kendi içinde sıralamak için, ders kapsamında işlenen ve projemizde verimli çalışan bir sıralama algoritması entegre ettik. Bir önceki uygulama gibi bu uygulamada da eşantiyon fotoğrafları okuyup filtrelenmiş fotoğrafı yazmak için pgm_read ve pgm_write fonksiyonlarını kaynak koda dahil ettik. Efekt fonksiyonlarında olduğu gibi eşantiyon fotoğrafları gezmek amacıyla döngüleri kullandık.

Ek olarak projemizdeki kaynak kodları tek adımda derlemek için bir makefile dosyası kullandık.

Efekt ve Filtreleme uygulamalarının kullanımları birbirlerinden çok farklı olsa da aynı makefile üzerinden derlenmektedirler. Unix tabanlı işletim sistemlerinin komut satırından make komutu ile çalıştırılmaktadır. İçeriğinde ise, ek olarak yazdığımız ve ana (main) programda kullanılan kütüphaneleri dahil etmesi istenir. Efekt uygulaması için dosya işlemi kütüphanesi ve efekt kütüphanesi; filtreleme uygulaması için sadece dosya işlemi kütüphanesini dahil eder.

Efekt uygulamasının kullanımı komut satırından gerçekleşmektedir. Bu program komut satırından aldığı argüman/parametrelerle çalışır. İlk aldığı argüman ./ işaretinden sonra gelen programın ismidir. Sonrasında uygulanacak efektin ismi ve hemen ardından da efektin uygulanacağı resim dosyalarının işletim sistemindeki tam adresleridir (full-path). Programa istenildiği kadar resim dosyası verilebilir.

Örnek kullanım:

```
./pgm_efekt <effect name:noise,threshold,mirror,turn 90>
<image 1> <image 2> ……. <image n>
```

Filtreleme programı ise herhangi bir parametre almaz. Derledikten sonra ./pgm_median komutuyla çalıştırabilir.(önemli not: filtrelenecek eşantiyon dosyalarının, programla aynı dizinde olması gereklidir.)

Proje kapsamında, elimize geçen kaynak kodun genel yapısında herhangi bir değişiklik yapmadık. Elimizde olan veri yapıları ve fonksiyon argümanları proje uygulanabilirliği açısından gayet yeterliydi.

Yaptığımız ufak değişiklikler, effects.h dosyasında önceden tanımlanmış smooth efektini değiştirmek, ve makefile dosyasına pgm_median derleme problemini çözecek bir kod satırı eklemek oldu.

Proje grubumuz 3 kişiden oluşup projeye başlanıldığından itibaren her hafta kodun bir kısmı tamamlanmıştır. Projeye başlanılan ilk hafta pgm.c dosyasının kodu yazıldı. İkinci hafta effects.c ve effect_main.c tamamlandı. Üçüncü hafta da median_main.c dosyasındaki kod yazıldı ve effects.c'e effect_mirror ve effect_turn_90 fonksiyonları eklendi.

Her hafta yapılan uygulamalar gruptaki herkes adına geçerliydi ve hafta sonunda zoom ya da google meet üzerinden yapılan görüşmelerde yazılan kodlar karşılaştırıldı. Herkesin aynı şeyi yapması konunun ve pseudo kodun anlaşılması açısından grubumuz adına önemliydi. En başta yapmaya karar verilen şey adım adım ve düzenli bir şekilde ilerlemekti. Kafa karışıklığına yol açmadan yapılacakları planlayarak ve deadlinelar belirleyerek süreç ilerledi.

Rapor yazma sırasında ise görev bölümü yapıldı. Raporun introduction, results ve conclusion kısımları grup üyeleri arasında bölüşüldü. Project architecture kısmı ise ortak bir şekilde yazıldı. Bu projenin yapımı kod kısmı 3 hafta rapor kısmı da 1 hafta olmak üzere yaklaşık 4 hafta sürmüştür. Dört hafta sonunda gönderilmeden önce kod ve raporun üstünden son bir kez daha geçildi değişiklik yapılması gereken yerler düzeltildi.

Proje kapsamında PGM türündeki fotoğraf dosyaları üzerinde çalıştık. Bu dosyaları açıp satır satır okuduk ve okuduklarımızı yeni bir PGM dosyasına yazdık. İstenen sayıda fotoğraf dosyasına 5 farklı efekt uygulamayı başardık. (Invert, Binarize, Noise, Turn ve Mirror) Bunun yanı sıra aynı açıdan çekilmiş birden fazla fotoğrafı kullanarak

“medyan filtreleme” yöntemi sayesinde istenmeyen detayları yok ettik.

Ancak kodlarımız sadece PGM dosyalarında uygulanabiliyor ve sadece Unix tabanlı işletim sistemlerinde kullanılabiliyor. Kodumuzda Linux kütüphaneleri aktif kullanıldığı için Windowsta kullanmak zorlaşıyor.

İskelet kod üzerine projemizi uygularken karşılaştığımız sorunlardan ilki pgm.c dosyasındaki pgm_read() ve pgm_write() fonksiyonlarını yazmak oldu. Dosyanın satırlarını düzgünce okumakta ve yazmakta zorlandık. Test amaçlı okuduğumuz her satırı ekrana basarak karşılaştırıp doğru olup olmadıklarını kontrol ettik.

Bu süreçte C kütüphaneleri taramak ve alternatif fonksiyonları bulup test etmek zorunda kaldık. Aldığımız hataları çözümleyebilmek için programımızı Eclipse Debugger programına soktuk.

Karşılaştığımız sorunlardan diğeri ise “median_main.c” dosyamızı çalıştıramıyor oluşumuzdu. Sorunun kaynağının makefile dosyasına olduğunu anladık ve gerekli düzenlemeleri yaptık.

Dosyamızda all: başlığı altında iki program dosyasını (pgm_median ve pgm_efekt) derlememiş olmamızdı.

```
CC=gcc
CFLAGS=-O2

DEPS = pgm.h effects.h

%.o: %.c $(DEPS)
	$(CC) $(CFLAGS) -c -o $@ $<

pgm_efekt: pgm.o effects.o effects_main.o
	$(CC) $(CFLAGS) -o $@ $^

pgm_median: pgm.o median_main.o
	$(CC) $(CFLAGS) -o $@ $^

clean:
	rm -rf *.o pgm_efekt pgm_median binarize_* noise_* invert_*
```

Makefile dosyamızın önceki hali

```
CC=gcc
CFLAGS=-O2

DEPS = pgm.h effects.h


%.o: %.c $(DEPS)
	$(CC) $(CFLAGS) -c -o $@ $<

all: pgm_median pgm_efekt

pgm_efekt: pgm.o effects.o effects_main.o
	$(CC) $(CFLAGS) -o $@ $^

pgm_median: pgm.o median_main.o
	$(CC) $(CFLAGS) -o $@ $^

clean:
	rm -rf *.o pgm_efekt pgm_median binarize_* noise_* invert_* smooth_* rotate_*
```

Düzenlenmiş hali.

PROJE SONUÇLARI

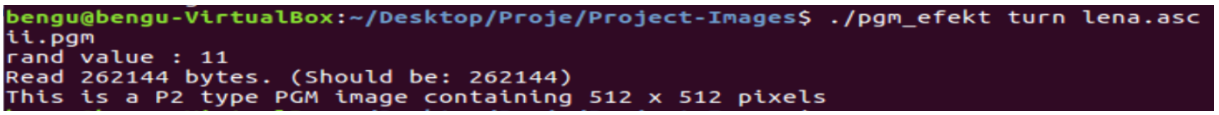
Makefile dosyası ile kaynak kodlar derlendiğinde şu dosyalar oluşmaktadır:

Oluşan Adları	Derlenen Dosya Adları
effects.o	effect.c effect.h
effects_main.o	effect_main.c effect.o pgm.o
median_main.o	median_main.c pgm.o
pgm.o	pgm.c pgm.h
pgm_efekt	effect_main.c effect.o pgm.o
pgm_median	pgm.o median_main.c

Sonu .o ile biten dosyalar, kaynak kodların derlenmesi sonucunda ortaya çıkmış dosyalardır. .o uzantılı object dosyalarıdır. Object dosyaları ismini içeren .c dosyalarının binary formlarıdır. Ana programımızda kullanacağımız fonksiyonları başka kaynak dosyalarında tanımlamış olabiliriz o halde bu tanımlanan fonksiyonları bir şekilde ana programımızla birleştirmeliyiz.

pgm_efekt ile pgm_median dosyaları ise .exe tipinde çalıştırılabilir dosyalardır. Terminal ekranından kodu çalıştırmak için kullanılırlar.

Eğer efekt uygulanacaksa terminal ekranından pgm_efekt dosyası, sırasıyla yapılması istenen efekt adı ve efekt uygulanacak dosyanın adı girilerek çalıştırılmalıdır. İşlem bittiğinde efekt yapılmış fotoğrafın adına, yapılan efektin adı eklenilerek oluşturulacaktır.



Efekt programının çalıştırılma örneği verilmiştir. Turn efekti, lena.ascii.pgm dosyamıza uygulanmıştır. Yeni çıkan dosyanın adı “lena.ascii.pgm.turn” olarak kaydedilmiştir.

Medyan filtreleme işlemi için ise terminal ekranımıza ./pgm_median yazmamız yeterlidir. Oluşan dosyamız “filtered.pgm” şeklinde adlandırılacaktır.

Proje boyunca aktif kullandığımız dosya okuma (pgm_read) ve dosya yazma (pgm_write) fonksiyonlarının yazıldığı pgm.c dosyasında, parametre olarak verilen fotoğrafın bilgileri okunurken fgets() değil fscanf() fonksiyonu tercih edildi. İki fonksiyon da stdio.h kütüphanesinde bulunmaktadır ve aşağı yukarı benzer işlevdedir. Projemizde hafıza kullanımı ve çalışma süresine önem verdiğimiz için iki fonksiyonu da deneyerek elde ettiğimiz sonuçları karşılaştırdık. Dosya okunurken en ve boy değerlerinin hesaplanmasında fgets() kullandığımızda hem en hem de boy bilgileri için ayrı döngüler kullanmamız gerekiyordu ve bu da $O(2N^2)$ -> $O(N^2)$ lik bir kayba sebep oluyordu. Ancak fscanf() kullanımı ile tek satırda bitirdik. Bu sebeple pgm.c dosyasında fscanf() fonksiyonu kullanıldı. (O(1))

```
/* TODO: En ve boyu oku */
buffer = fgets(line,LINE_MAX,pgm);

while (line[i]!=' '){

  en_boy[i] = line[i];
  i++;
}
pgm_info.width = atoi(en_boy);

i++;

while(line[i]!='\n'){
  en_boy[x] = line[i];
  i++;
  x++;
}

pgm_info.height = atoi(en_boy);
```

Dosyanın en ve boy bilgileri fgets() ile okunmuştur.

```
/* TODO: En ve boyu oku */
A.value_i = fscanf(fp,"%d %d\n",&(pgm_info.width), &(pgm_info.height));
```

Dosyanın en ve boy bilgileri fscanf() ile okunmuştur.

Medyan filtreleme işlemimizi yaparken median_main.c dosyasında aldığımız 9 farklı resmin her pikselini kendi aralarında karşılaştırmamız ve ortancayı geri döndürmemiz gerekiyor. Piksel başına 9 farklı dosyadaki pikseli sıralarken bir sıralama algoritması seçmek durumunda kaldık. Aşağıdaki tabloda verildiği üzere çalışma sürelerini karşılaştırarak bizim için en iyi olacağını düşündüğümüz algoritmayı seçtik.

Algoritma Adı	Çalışma Süresi	Best Case	Worst Case
Insertion Sort	0.031909	O(N)	$O(N^2)$
Shell Sort	0.043234	O(N)	-
Quick Sort	0.034660	NlogN	$O(N^2)$
Heap Sort	0.037646	NlogN	NlogN

Not : Bu testler Windows üzerine kurulan Ubuntu 16.04 sürümü sanal makineyle test edilmiş olup işlemci Intel Core i7-1065G7 CPU 1.30 GHz ve işletim sistemi de 64 bittir.

Öncelikle, Selection Sort algoritmasının Best Case durumu $O(N^2)$ olup elimizdeki bütün algoritmalardan yavaş olduğu için, Bubble Sort algoritmasının veri büyüdükçe yavaş çalışmasından dolayı ve Merge Sort algoritmasının da ekstra bellek harcamasından dolayı tercih edilmediğini belirtmeliyiz.

Tabloda görüldüğü üzere 4 algoritma arasında süre hesabı yapılmıştır. Süre hesabının yapıldığı kod örneği aşağıdadır;

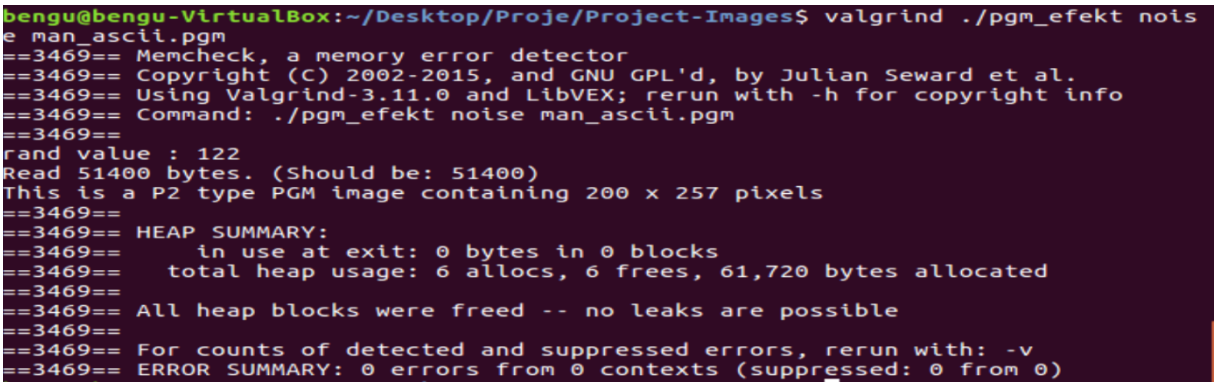
```
/* TODO: Median filtreleme */
    start_time = clock();  //zaman hesaplama
    for (k=0; k<size; k=k+1) {
        for (i=0; i<N_IMAGES; i=i+1) {
                ordered[i] = images[i].pixels[k];
        }
        filtered.pixels[k] = sort_and_get_median(ordered, N_IMAGES);
    }
    end_time = clock();
    printf("\nresult : %f\n",(double)(end_time-start_time)/CLOCKS_PER_SEC);
```

Zaman ölçümü için time.h kütüphanesinde yer alan clock() fonksiyonu kullanılmıştır. Sıralama algoritmamıza girmeden önce başlangıç zamanı ölçülmüş, sıralama algoritmamızın çıkışında ise bitiş zamanı ölçülmüş ve ikisi arasındaki fark alınmıştır. Bulunan sonuç CLOCKS_PER_SEC değerine bölünerek milisaniye cinsinden karşılığı hesaplanmıştır. Görebilmek için ekrana %.2f şeklinde

kısaltarak değil de %f şeklinde uzun haliyle basılmıştır. Çünkü %.2f olarak basınca hem Insertion Sort hem Quick Sort hem de Heap Sort algoritmalarının hızı 0,03 olarak yuvarlanıyor ve hangisinin büyük olduğu net görülemiyordu.

Tablodaki ölçümlere göre en hızlı algoritmamız Insertion Sort olduğu için medyan filtreleme konusunda tercihimiz o oldu. Quick Sort ve Heap Sort en hızlı algoritmalardan olsa da projemizde işlem yapılan veri sayısı yeterli olmadığı için öne geçememiştir. (Yeterli N sağlanırsa durumu gerçekleşmemiştir.)

Projemizde kodları yazarken herhangi bir memory leak hatası alıp almadığımızı kontrol etmek amacıyle Valgrind programından yararlandık. effect.c dosyası içine yazdığımız bütün efektleri ve median_main.c dosyamızı test ettik ve herhangi bir bellek sızıntısına rastlamadık. Malloc ile ayrılan belleklerin hepsinin program sonunda hatasız geri verildiği ortaya çıkmıştır.



Örnek bir Valgrind ölçümü. effects_main.c dosyası içinde bulunan noise efekti ölçülmüştür.

SONUÇ

Sonuç olarak, projeyi gerçekleştirirken çeşitli aşamalardan yararlandık. İlk aşama verilen problemin tanımı ile başlayıp, en uygun çözümün tasarlanması ile devam etti.

Efekt uygulamasındaki ana sorun, fotoğraf verisini okuyup,elde edilen bilgiyi, ulaşım kolaylığını da göz ardı etmeden saklamaktı. Bunun için fotoğrafın bütün özelliklerini (yorum, fotoğrafın en ve boy bilgileri, pixel bilgileri) barındıran ve üzerinde kolayca işlem yapabileceğimiz bir veri yapısı tasarladık. Bir sonraki adımda, dosya bilgilerini okumamıza yarayacak ve gerektiğinde elindeki bilgileri yeni bir dosyaya aktaracak yazılımı geliştirmek oldu.

Fotoğraf verisini okuyup depolandıktan sonra geriye istenen efekti uygulamak kaldı. Bu noktada kullandığımız veri tipi fotoğrafın pixellerine erişim olanağı sunuyordu. Fotoğraf üzerinde istenen efekti göz önünde bulundurarak, pixelleri yeniden düzenleyip yeni resmi, yeni bir dosyaya yazdık. Bu işlem sonucunda efekt uygulanmış resmi elde ettik. Ayrıca programımızın mimarisini birden fazla fotoğraf üzerinde işlem yapabilecek şekilde tasarladık. Böylelikle aynı efekti birden fazla fotoğrafa uygulamak istendiğinde, programı sadece bir kere çalıştırmak yeterli oldu. Böylelikle birbirinden farklı, aynı efekte sahip fotoğraflar elde ettik.

Projemizin ikinci adımı aynı açıdan çekilmiş dokuz fotoğraftan bir öğeyi silmek, filtrelenmiş bir fotoğraf elde etmekti. Bunu başarabilmek için median filtreleme(cf. Proje Mimarisi) tekniğine başvurduk. Median filtreleme uygulaması dokuz fotoğrafın her birinin pixel bilgisini ve bu bilgilerin sıralanmasını gerektiriyordu. Fotoğraflarımızın bilgilerini efekt uygulamasında da kullandığımız prensiple okuduk ve depoladık. Sonrasında her bir fotoğrafın, aynı pixel değerlerini sıralayıp filtre değeri yeni bir dosyaya yazdık.

Tercih edeceğimiz sıralama algoritması, programın verimliliği açısından büyük önem taşıyordu. Teorik bilgimize ve ölçümlere dayanarak en hızlı çalışan sıralama algoritmasını seçtik. Programı çalıştırdığımızda başarılı bir şekilde filtrelenmiş resmi elde ettik.

Günümüzde pgm veri tipi pek kalmadığından, daha kompleks yapıdaki fotoğraf dosyalarına uygun olan efekt programları geliştirilmiştir. Modern görüntü işleme teknolojisinde birçok efekt ve filtre uygulama yöntemi vardır. Median filtreleme de bunlardan biridir fakat buna ek olarak, Mean, Gaussian Smoothing , Conservative Smoothing, Frequency Filters gibi daha karmaşık efekt işlemleri kullanılmaktadır. Efekt uygulanacak resmin veri yapısı bizim projemizde olduğu gibi matris biçimindedir tek fark daha karmaşık, çok boyutlu ifade edilmesidir. Efekt teknikleri ise bu karmaşık görüntü yapılarında işlem yapabilecek şekilde tasarlanmıştır.

KAYNAKÇA

- ☐ http://www.ckutuphanesi.com/
- ☐ https://www.geeksforgeeks.org/
- ☐ https://tr.wikipedia.org/wiki/G%C3%B6r%C3%BCnt%C3%BC_i%C5%9Fleme
- ☐ https://tr.wikipedia.org/wiki/S%C4%B1ralama_algoritmas%C4%B1#S%C4%B1ralama_Algoritmalar%C4%B1
- ☐ INF103 Algoritma ve İleri Programlama Ders Notları