Deniz Özkan
200309027

Projemde bir görüntüyü kolon bazında birçok parçaya bölerek histogram eşitleme işlemi yaptım. Histogram eşitleme görseldeki kontrastı artırarak nesnelerin daha belirgin olmasını sağlayan bir görüntü işleme tekniğidir. Görselin bölümlenmiş her bir parçası başlangıç ve bitiş kolonuna göre bir thread'e atanarak eşitleme işlemine sokulmuştur.

İlk versiyonda görsel herhangi bir parçalama işlemine tabi tutulmadan histogram eşitleme yapılmıştır. Görselin her bir pikselinden elde edilen renk derinliği ilgili fonksiyonlar yardımıyla azaltılarak veya artırılarak eşitlenmiştir.

Multithreadli versiyonda ise programa verilen görsel parçalara bölünmüştür ve her bir parçanın sınırları belirlenmiştir. Bahsedilen sınırların başlangıç ve bitişi görseldeki belirli kolonlara karşılık gelmektedir. Programa argüman olarak geçilen thread sayısı sayesinde görsel bu thread sayısı kadar eş parçaya bölünerek her bir parça için threadle işlem yapılmıştır.

Aşağıdaki Image sınıfında bir görseli eşitleme işlemine hazır hale getirmek için bir dizi işlem yapılmaktadır. Yine bu sınıfta üzerinde oynama yapılan görseli ve histogramını ekranda göstermek için display fonksiyonu tanımlanmıştır.

```
//Görselle ilgili ilk işlemler ve görseli ekrana verme için sınıf tanımlanmıştır.
//Histogram initalize etme ve hesaplaması için yardımcı fonksiyonlar bu sınıf içindedir.
class Image {
private:
   //İşlem yapılan görseller ve histogramları için mat tipinden tanımlamalar yapıldı.
   cv::Mat original, output, histGraph, histGraphOutput;
   //histogram derinliğinden dizi boyutu belirlendi
   double histogram[DEPTH];

   //dizinin bütün değerleri 0'la doldurulduu
   void initializeHist() {
      for(int i=0; i<DEPTH; i++)
         histogram[i] = 0;
   }

   //verilen görselin satır ve sütununda gezildi ve histogram dizine atama yapıldı
   void histHesapla() {
      for(int i=0; i<original.cols; i++)
         for(int j=0; j<original.rows; j++)
            histogram[original.at<uchar>(i, j)]++;
   }

   //renkler üzerinde normalizasyon yapıldı ve ilgili dizi indexine tekrar atama yapıldı.
   void histogramEsitleHesapla() {
      int toplam = original.rows * original.cols;

      for(int i=1; i<DEPTH; i++)
         histogram[i] = histogram[i-1] + histogram[i];

      for(int i=0; i<DEPTH; i++) {
         histogram[i] = (histogram[i] / toplam) * (DEPTH - 1);
         histogram[i] = (int) (histogram[i] + (0.5));
      }
   }

      //ilgili görseli ekranda göstermek için fonksiyon
      void display(Mat &image, string title) {
      namedWindow(title, WINDOW_AUTOSIZE);
      imshow(title, image);

   }
```

Öncelikle threadsiz haliyle alttaki iki fonksiyonlar aracılığıyla eşitleme işlemi yapılmaktadır. Alttaki fonksiyonda image sınıfından ilgili fonksiyonlar çağrılarak eşitleme işlemi başlatılmıştır.

```
void Image :: histogramHesapla() {
   //histogram hesaplama fonksiyonu çağrılır
   histHesapla();

   //histogram eşitleme hesaplaması fonksiyonu çağrılır
   histogramEsitleHesapla();

   cout << "\n\nGörsel Boyutu:\nYükseklik = " << original.rows << endl;
   cout << "Genişlik = " << original.cols << endl << endl;

   clock_t baslangicZamani = clock();
   histogramEsitle(this);

   //çalışma süresini gösteren kod
   cout << fixed << setprecision(11);
   cout << "\nÇalışma Süresi: " << (double( clock() - baslangicZamani ) / (double)CLOCKS_PER_SEC) / 1000 << " saniye." << endl;


   //tüm işlemler sonucu oluşan görüntüler ekrana verilir.
   display(output, "Histogram Eşitlenmiş Görüntü");

   histGraphOutput = calculateHistogramImage(output);
   display(histGraphOutput, "Son Histogram Hali");
}
```

Altta verilen fonksiyon üstte verilen kodun içinde çağrılarak eşitleme işlemi tamamlanmıştır ve görseller ekrana verilmiştir.

```
//histogram eşitleme fonksiyonu
void *histogramEsitle(Image *img) {


   for(int i=0; i< img->output.cols; i++)
      for(int j=0; j<img->output.cols; j++)
         img->output.at<uchar>(i, j) = img->histogram[img->original.at<uchar>(i, j)];
   return img;
}
```

Programın bir sonraki versiyonunda ise bahsedildiği gibi ilgili görsel multithread ile işlenerek performans artışı beklenmiştir. Altta verilen fonksiyonda istenen sayıda thread oluşturma ve o sayıya göre görseli parçalama işlemi yapılmaktadır. Dikey olarak bölümlenmiş görselde her bir parça için bir thread oluşturularak ilgili fonksiyonda eşitleme işlemi yapılmaktadır.

```
void Image :: histogramEsitle(int threadSayisi) {
   //histogram hesaplama fonksiyonu çağrılır
   histHesapla();

   //histogram eşitleme hesaplaması fonksiyonu çağrılır
   histogramEsitleHesapla();

   //thread sayısına göre resmin kaç parçaya bölüneceğini ve bölünen parçalara başlangıç ve bitiş değeri ataması yapan kod parçası
   if(original.cols < threadSayisi)
      threadSayisi = original.cols;

   pthread_t Thread[threadSayisi]; //elle girilecek thread sayısı bu diziye atanır

   //adım sayısını hesaplama
   int adimSayisi = original.cols / threadSayisi;
   int basla, bitir;

   cout << "\n\nGörsel Boyutu:\nYükseklik = " << original.rows << endl;
   cout << "Genişlik = " << original.cols << endl << endl;

   clock_t baslangicZamani = clock();
   for(int i=0; i<threadSayisi; i++) {
      basla = (i * adimSayisi);

      if(i == threadSayisi)
      bitir = original.cols;
      else bitir = basla + adimSayisi;

      cout << "Thread numarası: " << i << "\tBaslangıç Pikseli = " << basla << "\t Bitiş Pikseli = " << bitir << endl;

      // görselin bölündüğü parçalar için elle girilen thread sayısı kadar thread yaratılır.
      pthread_create(&Thread[i], NULL, histogramEsitleThread, (new GorselSinirlari(basla, bitir, this)));
   }

   //yaratılan threadler join ile bağlanır
   for(int i=0; i<threadSayisi; i++)
      pthread_join(Thread[i], NULL);
```

Görselin sınırlarının belirlendiği sınıf ve bu sınıftan yaratılan nesne sayesinde eşitleme işlemi yapan fonksiyon aşağıdaki kodda verilmiştir.

```
//multithread için parçalanmış görselin sınırlarını belirleyen sınıf
class GorselSinirlari {
private:
   int basla;
   int bitir;
   Image *img;

//constructor ve yaratılacak nesnein bu sınıfın elemanlarına erişmesi için friend constructor
public:
   GorselSinirlari(int Basla, int Bitir, Image *Img) {
      basla = Basla;
      bitir  = Bitir;
      img = Img;
   }
   friend void *histogramEsitleThread(void*);
};


//her bir bölünmüş parça için eşitleme yapan fonksiyon
void *histogramEsitleThread(void *sinirlar) {

   //bölünmüş parçalara erişmek için ilgili sınıftan nesne yaratılmıştır.
   GorselSinirlari *s = static_cast<GorselSinirlari *>(sinirlar);

   //histogram eşitleme işlemini yapan döngüler
   for(int i=0; i<s->img->output.cols; i++)
      for(int j=s->basla; j<s->bitir; j++)
         s->img->output.at<uchar>(i, j) = s->img->histogram[s->img->original.at<uchar>(i, j)];
   return sinirlar;
}
```

Ayrıca derleme esnasında -O3 komutuyla yapılan optimizasyon sonucu çalışma süresine olumlu katkıda bulunmuştur ve çalışma süresini hem threadli versiyonda hem de threadsiz versiyonda neredeyse dört kat kısaltmıştır.

Aşağıdaki karşılaştırmada -O3'süz ve -O3 ile derleme sonucu oluşan fark gözlemlenmektedir.

-O3 olmadan yapılan derleme sonucu:

```
(base) deniz@deniz:~/Desktop/histogramEsitle$ ./histogramEsitlemeMulti 1.jpg 2


Görsel Boyutu:
Yükseklik = 512
Genişlik = 512

Thread numarası: 0      Baslangıç Pikseli = 0    Bitiş Pikseli = 256
Thread numarası: 1      Baslangıç Pikseli = 256  Bitiş Pikseli = 512

Çalışma Süresi: 0.00000216600 saniye.
```

-O3 kullanılarak yapılan derleme sonucu:

```
(base) deniz@deniz:~/Desktop/histogramEsitle$ ./histogramEsitlemeMulti 1.jpg 2


Görsel Boyutu:
Yükseklik = 512
Genişlik = 512

Thread numarası: 0      Baslangıç Pikseli = 0    Bitiş Pikseli = 256
Thread numarası: 1      Baslangıç Pikseli = 256  Bitiş Pikseli = 512

Çalışma Süresi: 0.00000056000 saniye.
```

Karşılaşılan Problemler

*Programlamayı java ile öğrendiğim için pointer, referencing, dereferencing mantığını kafamda oturtmam biraz zaman aldı. C++'ta bir objenin adresindeki verinin nasıl manipüle edilebileceğini öğrendim. Pointerlar burada bana yardımcı oldu.

*Üzerinde işlem yaptığım görsel değişkenlerini private tanımlayarak onları olası bir müdahaleye veya bozukluğa karşı korumuş oldum fakat private olarak tanımlanmış değişkenlere ve fonksiyonlara erişebilme yollarını öğrendim.

*Yazdığım programda görsel okuma işlemi için linux üzerinde kullandığım opencv kütüphanesini tanımlamak zaman alıcı oldu. Opencv'nin farklı versiyonlarda farklı fonksiyon isimlerine sahip olması görsel okuma işlemi için harcadığım zamanı olumsuz yönde etkiledi. Versiyon bazında araştırma yaparak hangi fonksiyonun kullanılması gerektiğini öğrendim.

*Threadleri öğrenirken karşılaştığım problemlerden biri join yapısı oldu. Detach ve join farkını yaptığım araştırmalara sonucu öğrenmiş oldum. Ben de projemde join fonksiyonunu uygulamış oldum.

*Single thread ve multi thread olarak iki versiyona sahip programda görselleri threadlere atayarak yaptığım işlem sonunda beklediğim performans artışı gözle görülür seviyede olmadı. En belirgin fark 1 ve 2 thread arasında oldu. Program 2 threade bölündüğünde 1 threadden daha fazla performans vermesine karşın 2'den fazla yaratılan thread performans kaybına neden oldu. Çünkü üzerinde işlem yapılan görseller görece küçük olduğu için onları parçalamak ve yeni threadler yaratmak programa yük bindirmiş oldu. Bu yüzden 2 thread ile optimum performansı almış oldum.