

**KARELER ARASI  
DEĐİŐİMİN BELİRLENİP TELAFİ  
EDİLMESİ YÖNTEMİ İLE  
HAREKETLİ GÖRÜNTÜ  
SIKIŐTIRILMASI**

**A. AYLİN TOKUÇ**

**ÖZEL EGE LİSESİ  
İZMİR - 2001**

## **TEŐEKKÜR**

Projenin konusu için fikir veren ve algoritmayı kurmama yardımcı olan, çalışma esnasında fikirleriyle takıldığım noktaları aşmamı sağlayan Ege Üniversitesi Network Yönetim Kurulu Üyesi Sayın Muhammed CİNSDİKİCİ 'ye teşekkürü borç bilirim.

Ayrıca proje çalışmamı takip edip bana destek olan rehber öğretmenim Sayın Selvinaz GÖKALP 'e, Özel Ege Lisesi bünyesinde çalışan bütün idareci ve öğretmenlerime, moral desteğini esirgemeyen arkadaşlarıma, ve yıllardır yanımda olup bana uygun çalışma ortamını sağlayan aileme teşekkürlerimi sunarım.

## İÇİNDEKİLER

|                                |    |
|--------------------------------|----|
| A. Giriş ve Amaç.....          | 1  |
| B. Algoritma.....              | 3  |
| C. Gerçekleştirme Bilgisi..... | 5  |
| 1. Programın Kodlanması.....   | 5  |
| 2. Dosya Yapısı.....           | 6  |
| D. Sonuçlar ve Tartışma.....   | 7  |
| E. Araç Gereçler.....          | 14 |
| F. Kaynakça .....              | 14 |

## A. GİRİŞ VE AMAÇ

Veri sıkıştırmanın günlük hayata dahi ne denli girdiğine ve hayatımızı nasıl kolaylaştırdığına basit bir örnek vererek başlayalım:

Faks çekmek günümüzde her işyerinin her gün yaptığı rutin işlerdendir. Ama faks makinesi yolladığınız resmi olduğu gibi göndermez, göndermeden önce sıkıştırır. Bunun bize iki yararı olur: Eğer faks makineleri herhangi bir veri sıkıştırma tekniği kullanmasaydı faksımız on (veya daha fazla) kat uzun sürede giderdi, ki bu da telefon hattının on kat daha fazla meşgul edilmesi, ve on kat daha fazla para ödememiz demek olurdu. Bir de göz ardı edilmemesi gerekir ki bu on kat fazla zaman içinde telefon hattında bir parazitin oluşması ve veri kaybına neden olması ihtimali de bir o kadar artmaktadır.

Faks makinesi gibi pek çok örnek bugün hayatımıza girmiş, işlerimizi daha kısa zamanda halletmemizi ve daha ucuza mâl etmemizi sağlamaktadır. Örneğin bilgisayarınızdaki verilerinizin yedeklerini nasıl saklıyorsunuz? E-mail'le yollamak istediğiniz dosyaları nasıl ekliyorsunuz? Muhtemelen zip veya benzeri bir sıkıştırma aracı kullanıyorsunuz. Peki televizyon seyrederken? Dijital yayın yapan televizyon paketleri bu teknolojiyi kullanarak daha fazla istasyonu tek transporter üzerinden yayınlatabilmekteler. . .

Veri sıkıştırma yöntemlerini kabaca iki gruba ayırabiliriz:

### Kayıplı Sıkıştırma :

Kayıplı sıkıştırma yöntemleri özellikle multimedya verilerinde kullanılmakta. Buna son günlerde çok moda olan bir mpeg standardı olan mp3 formatı örnek verilebilir. Bu yöntemlerde veri geri açıldığında fark edilemeyecek kadar kayıp vardır. Mesela mp3 formatında insan kulağının duyamayacağı 10hz altı ve 44khz üzeri sesler kayıt edilmez. Bu sesleri duymadığımızı göre kaydetmenin de manası yoktur.

Aynı şekilde kayıplı veri sıkıştırma yöntemlerine jpeg standardı verilebilir. Bu yöntemde renk tonları arasındaki gözün göremeyeceği ayrıntılar silinir. Mesela siyah bir zemindeki siyah-gri arası tonlar silinir, bu ayrıntı resme ancak çok yakından bakıldığında seçilebilir.

### Kayıpsız Sıkıştırma :

Kayıpsız sıkıştırma yöntemleri daha çok sayısal sonuçların önemli olduğu dosyalarda kullanılır. Örneğin bir exe dosyasında bir kaç byte 'ın farklı olması sistemin kilitlenmesine veya exe 'nin işlem yapamamasına neden olabilir.

Bir video görüntüsünün boyutunu ve sıkıştırılabilirliğini etkileyen başlıca faktörleri kısaca üç ana başlık altında toplayabiliriz:

Resmin çözünürlüğü(boyutu):

Resmin ekranda görüntülenirken kapladığı alan. Resmin boyutu ne kadar küçük olursa dosyanın kapladığı alan da o derece küçük olacaktır. Ancak boyutun çok küçük olması durumunda ekranda neler olup bittiğini izlemek de güçleşecektir.

Resmin renk sayısı:

Her karenin sahip olduğu renk sayısı. Eğer 320x320lik bir video karemizin olduğunu düşünürsek bu karenin boyutu gerçek renk seçeneği için 320x320x24 bit olurken siyah beyaz renk seçeneğinde bir tek kare boyutunun 1/3'üne, yani 320x320x8 bit 'e inecektir.

Videoda karelerin sıklığı:

Video görüntüleri resimlerin art arda sıralanmasıyla elde edilir. Aslında ekrandaki şeyler hareket etmez, sadece hareket edermiş gibi görünür.

Bir saniyede 30 karenin ekrana yansıtıldığını düşünürsek sıkıştırılmamış bir görüntüde 320x320 piksellik boyutta 256 renk seçeneğinde 1 saniyelik bir video görüntüsünün boyutu 320x320x8x30 bit, yani yaklaşık 3 MB olur. Bu da demektir ki hiç sıkıştırılmamış, az önce söylediğimiz özelliklere sahip bir video görüntüsüyle VCD yapacak olsaydık bir CD'ye ancak 220 saniyelik, yani 4 dakikayı bulmayan veri kaydedebilirdik ki bir film için kaç CD harcanacağını boş verirsek 4 dakikada bir CD'yi değiştirerek film seyretmek hiç de zevkli olmazdı.

Bu projede bir video görüntülerinin tamamının hareket halinde olmaması esas alınarak sadece hareketli kısımların verilerinin depolanmasına dayalı bir sıkıştırmanın olabilirliği fikri ortaya atılmış, ve bu fikre uygun algoritma geliştirilip algoritmaya uygun bir program yazılarak algoritmanın uygulanması ve başarısının ölçülmesi amaçlanmıştır.

## B. ALGORİTMA

Daha önce de belirttiğimiz gibi her video görüntüsü art arda gelmiş resimlerden oluşur. Bu resimlerin oluşturduğu kareler plan değişmediği sürece tamamen değişmezler. Projede bu temelin üzerine bir algoritma kuruldu. Algoritma kısaca şöyle özetlenebilir:

Kareler bloklara ayrılır ve ardışık karelerin eş koordinatlı blokları arasında kıyaslamalar yapılır. Eş koordinatlı blokların içeriğinin farklı olması değişme, aynı olması ise değişmeme olarak adlandırılır.

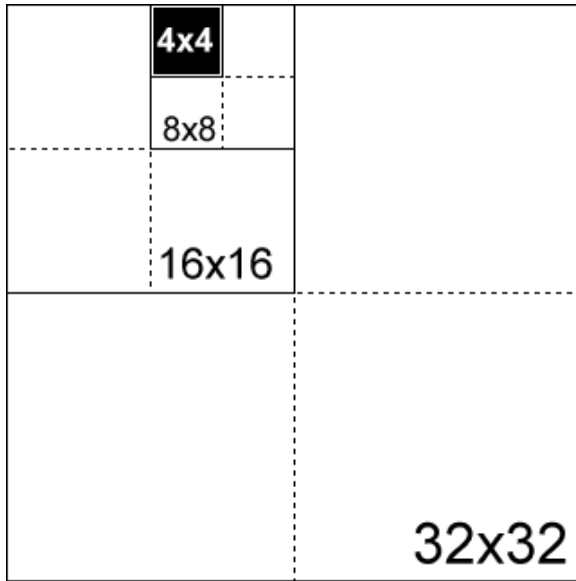
- Karelerdeki değişmeyen blokları yeni kare için tekrar kaydetmeye gerek yoktur.
- Değişmiş bir blok önceki karede farklı bir koordinatta bulunabilir(kaymış olabilir).

Algoritmayı daha ayrıntılı biçimde adım adım açıklayalım:

### Adım 1:

İlk kare renk haritasıyla birlikte kaydedilir.

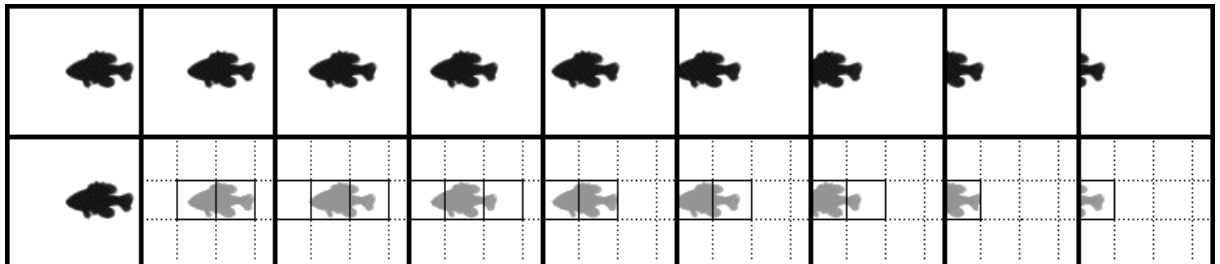
İlk adım bir kereye mahsus olarak gerçekleştirilir. Diğer adımlar ise video görüntüsünün son karesine kadar tekrar tekrar uygulanır.



### Adım 2:

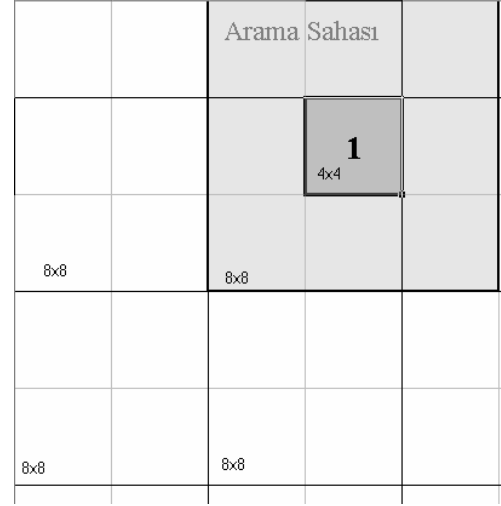
Video görüntüsünün kareleri  $n \times n$ 'lik bloklara bölünür.  $n$ 'in ilk değeri 32 iken bloklar  $n=4$  olana kadar recursive (öz yinelenmeli) olarak 4'e bölünür.  $4 \times 4$ 'lük eş koordinatlı bloklar  $t_i$  ve  $t_{i+1}$ . kareler için karşılaştırılır. Eğer bloklar arasında değişme yok ise (içerikleri aynı ise) blok 0 ile, değişme var ise (içerikleri farklı ise) 1 ile işaretlenir. Bloklar değişmemiş ise (0 ile işaretlenmişse)  $t_{i+1}$ . karenin bu bloğu için veri saklamaya gerek yoktur, çünkü bu bloğun içeriği ile ilgili bilgi zaten  $t_i$ . blok için kaydedilmiştir.

Blokları recursive olarak büyükten küçüğe oluşturup, işaretlemeyi küçükten büyüğe yaparak tüm matrisi bir kerede dolaşp bütün işaretlemeleri yapmış oluruz. 8x8lik bir karenin içinde bir hareket yakalayıp daha sonra o kareyi 4x4lük karelere bölmektense, 4x4lük bir karede hareket yakalanınca onun parçası olduğu 8x8lik karenin diğer parçalarının da hareket edip etmediğini görmek, ve gerekirse bu 4x4lük kareleri tekrar 8x8lik bir kare haline getirip 8x8liği 1 olarak işaretlemek zaman ve işlem tasarrufu sağlar.



### Adım 3:

Bir blok 1 (değişmiş) olarak işaretlenmiş olsa da bu blok için veri saklamaya gerek olmayabilir. Blokların 1 olarak işaretlenmesinin sebebinin bir yer değiştirme olması, bir cismin birdenbire ortada belirmesinden daha muhtemeldir. Ve eğer görüntüde yer değiştirmekte olan bir cisim varsa bu cismin bir önceki karedeki konumundan fazla uzaklaşmamış olması olasıdır. Bunu düşünerek 1 olarak işaretlenen  $n \times n$ 'lik bir blok  $3n \times 3n$ 'lik bir alanda aranır. Aranma sahasını bloğun etrafına kendi boyutunda bir çerçeve çizilmiş gibi düşünebiliriz. (Şekilde 1 ile işaretlenmiş koyu renkli alanı aranacak blok olarak düşünürsek açık gri alan arama alanıdır.)



### Adım 4:

Arama sonucunda bir önceki karede bloğun içeriğine sahip farklı koordinatlı bir blok bulunursa bu bloğun koordinatlarının saklanması yeterlidir. Ancak böyle bir blok bulunamaz ise bloğun yeni içeriği kaydedilir.

Görüntü karelerinde 1 ile işaretlenen alanlar arama sonucunda bulunursa “top left a dg(1) X Y”, bulunamazsa “top left a dg(0) Z” şeklinde paketler çıktı dosyasına eklenir. Z,  $a \times a$  'lık bir matrisin içeriğini belirtmektedir.

Bir kare ile ilgili işler bittiğinde yeni kareye geçildiği anlamında 0 yazılır. Yeni karede de Adım 2'den itibaren anlatılan basamaklar tekrarlanır.

## C. GERÇEKLEŞTİRME BİLGİSİ

### 1. PROGRAMIN KODLANMASI

Yukarıda anlatılan algoritmaya uygun kaynak kod MATLAB programında yazıldı. MATLAB programında kodlama yapılırken aşağıda ayrıntılı olarak açıklanan .m uzantılı fonksiyonlar yazıldı.

#### vcomp.m:

Diğer fonksiyonları çağıran ana fonksiyondur. İlk karenin renk haritasını ve içeriğini çıktı dosyasına kaydedip boyut ve renklerin yazılacağı formatı hesaplar (bakınız veri yapısı, \* ve \*\*). s. kare için s+1. karenin var olması durumunda koşulu sağlanan bir while döngüsü içinde ardışık iki karenin eş koordinatlı noktalarındaki renk değerlerinin farklarını saklayan bir matris yaratır (C matrisi) div32 fonksiyonunu çağırır.

#### div32.m:

C matrisini 32x32'lik bloklara böler. Her blok için div fonksiyonu çağırılır.

#### div.m:

top, left, ve a olmak üzere üç değişken alır. top ve left bloğun sol üst köşesini belirtirken a bloğun bir kenarıdır. a 'nın 4 'ten büyük olması halinde blok 4 'e bölünerek 4 tane div fonksiyonu çağırır. Fonksiyon arama işleminin sonucu olan 0 veya 1 sayılarını döndürür (değişim varsa 1, yoksa 0). Eğer a değeri 4'e ulaşmışsa tara fonksiyonu çağırılır.

Her 4x4lük blok kendisi için bir değer döndüğünde (0 veya 1) eğer aynı 8x8lik bloğa ait alt blokların hepsi aynı değeri döndürür ise 8x8lik blok da bu değeri döndürür. Recursive olarak açılan fonksiyon kapanırken çağırdığı 4 div farklı değerler döndürürler ise 1 dönenler için ara fonksiyonu çağırılır.

#### tara.m:

top ve left değerlerini alarak 4x4'lük bir bloğun değişip değişmediğini C matrisine bakarak bulur. Değişme varsa 1 yoksa 0 döndürür.

#### ara.m:

top, left, a olmak üzere 3 değer alır ve bu 3 değerın işlevleri div fonksiyonundaki gibidir. Arama sahasını belirler ve bloğu bu saha içerisinde arar. Arama sonucunda başarı sağlansa da sağlanmasa da yaz fonksiyonu çağırılır.

#### yaz.m:

top, left, a, x, y olmak üzere 5 değişken alır. top, left ve a 'nın işlevi aynıdır. x ve y değişkenleri arama sonucunda başarı sağlanmış ise bloğun bir önceki karedeki koordinatlarını, arama bir sonuç vermediyse 0, 0 değerlerini içerir.

#### decomp.m:

vcomp dosyasının çıktısından veri okuyup .bmp formatlı kareler oluşturan fonksiyondur. Aşağıda açıklanan dosya yapısına uygun olarak dosyadan bitler halinde okur.

## 2. DOSYA YAPISI

.bmp formatındaki video görüntü karelerinin yukarıda anlatılan algoritmaya uygun olarak sıkıştırılmasıyla elde edilen dosyanın içeriği şöyledir:

karelerin boyutları(piksel olarak) ..... 10'ar bit(X ve Y) unsigned integer  
bmp 'nin renk sayısı.....8 bit(MX) unsigned integer  
bmp 'nin renk haritası..... MX x 3'lük matris float(32 bit)  
ilk .bmp 'nin içeriği..... X x Y matrisi unsigned integer(\*)  
sonraki bmp 'nin değişen kısımları:

Değişen kısımlar top left a dg ..... şeklinde kaydedilirler.

**top** : değişen bloğun en üst satırının numarası(\*\*)

**left** : değişen bloğun en sol sütununun numarası(\*\*)

**a** : değişen bloğun boyutu (a x a)

**dg** : 1 bitlik unsigned integer, 1 ise blok kaymış, 0 ise değişmiş.

dg=0 ise paketin devamında a x a tane unsigned integer bulunur.(\*)

dg=1 ise paketin devamında 2 tane unsigned integer(\*\*) bulunur, bunlar bloğun bir önceki karedeki koordinatlarını belirtirler.

bir sonraki kare işareti:

*top* değeri olarak 0 okunursa bundan sonraki bilgilerin bir sonraki kareye ait olacağı anlaşılır(matris boyutları 0'dan başlamamakta, 1'den başlamaktadır. Yani sol üst köşesinin koordinatı (1,1) 'dir. Bu durumsa *top* değerinin 0 olması imkansızdır).

---

(\*) unsigned integerin bit sayısı bmp 'nin renk sayısına göre değişir. 32 renk için 5 bit, 256 renk için 8 bit vb.

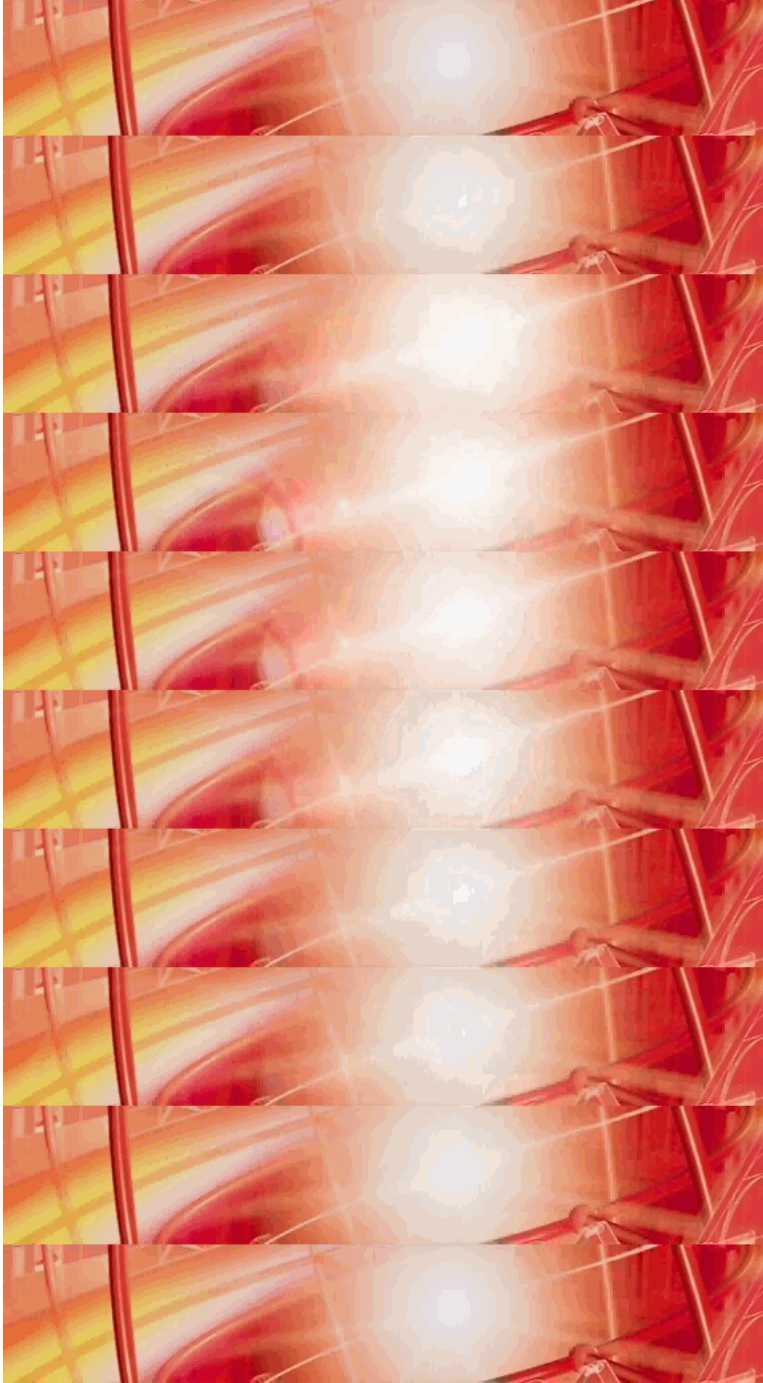
(\*\*) karelerin boyutu göz önüne alınarak gereken minimum bit sayısı hesaplanır ve o kullanılır.

## D. SONUÇLAR VE TARTIŞMA

Programın kodlanması tamamlandıktan sonra algoritmanın başarısını denemek amacı ile video görüntülerinden kaynak .bmp dosyaları oluşturulmuştur. Bu iş için Gamani GIF Movie Gear programından yararlanılmıştır.

GIF Movie Gear programı ile açılan .avi dosyalarının her karesinden inp#.bmp adlı ayrı dosyalar oluşturulmuştur. (# işareti kaçınıcı kare olduğunu belirten sıra numarasıdır)

Sonuçlara sayısal örneklerle yaklaşalım:



Sol tarafta bazı kareleri alt alta dizilmiş olarak görülen görüntü toplam 61 kareden oluşmaktadır. Bmp dosyaları 252 renkli ve 576x104 piksel boyutundadır.

bmp dosyalarının toplam boyutu :  
3 719 902 bayt

Çıktı dosyasının boyutu:  
1 684 233 bayt























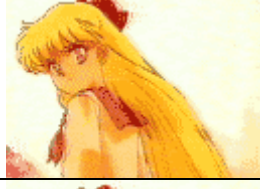
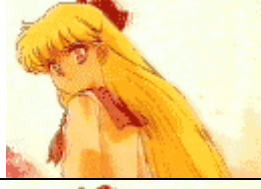












Sıkıştırma oranı:  
**% 54,7**

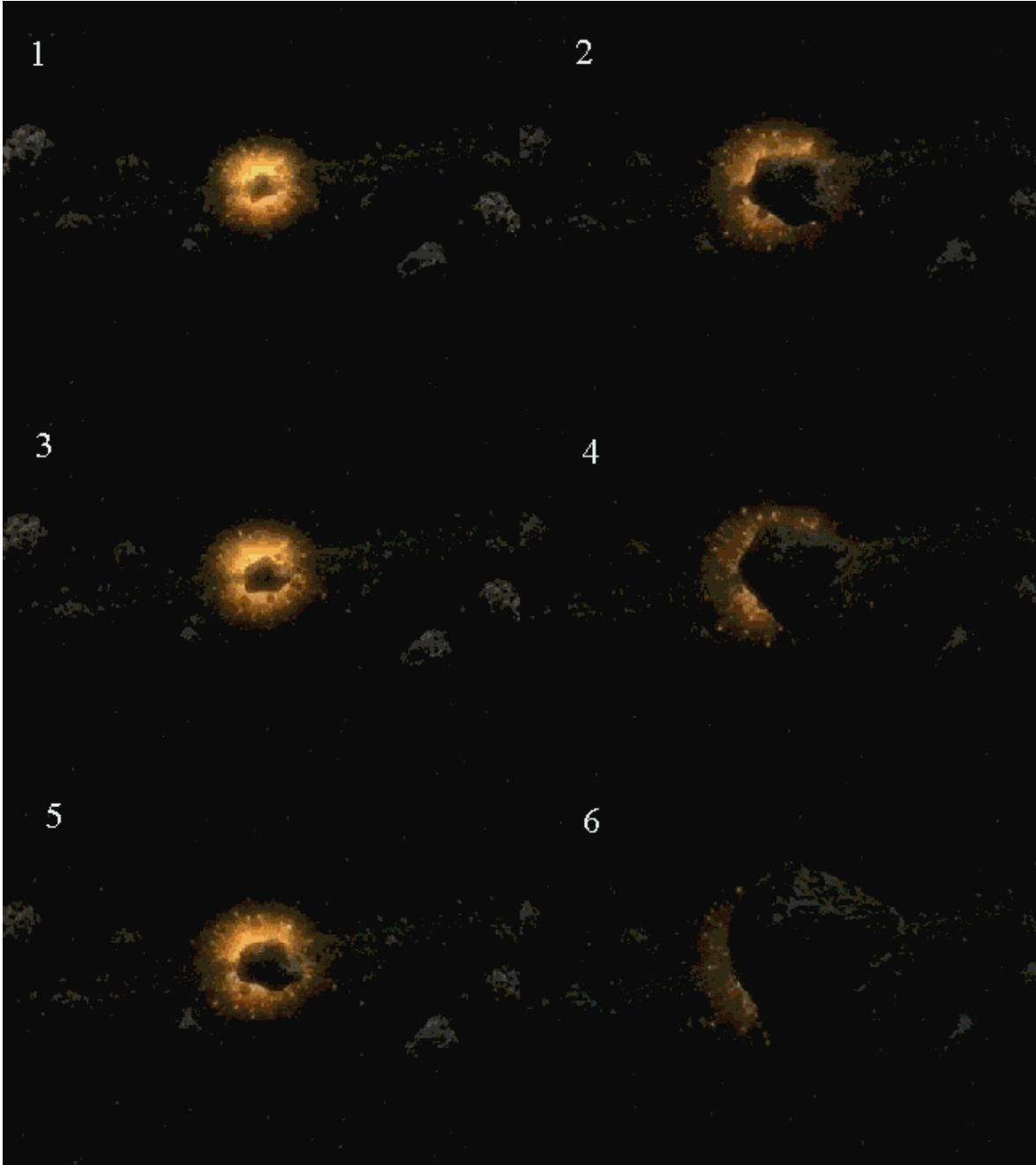
8. sayfadaki tabloda görülen, 32 renkli 129x115 piksellik 19 kareden oluşan görüntünü için:

bmp dosyalarının toplam boyutu :  
308 902 bayt

Çıktı dosyasının boyutu:  
202 160 bayt

Sıkıştırma oranı:  
**% 34,5**

|   | girdi   | çıktı   |    | girdi  | çıktı   |
|---|---|---|----|--|---|
| 1 |    |    | 10 |    |    |
| 2 |    |    | 11 |    |    |
| 3 |    |    | 12 |    |    |
| 4 |   |   | 13 |   |   |
| 5 |  |  | 14 |  |  |
| 6 |  |  | 15 |  |  |
| 7 |  |  | 16 |  |  |
| 8 |  |  | 17 |  |  |
| 9 |  |  | 18 |  |  |



6 karesi yukarıda görülen ve 256 renkli 320x240 piksellik 34 kareden oluşan video görüntünün:

bmp dosyalarının toplam boyutu:

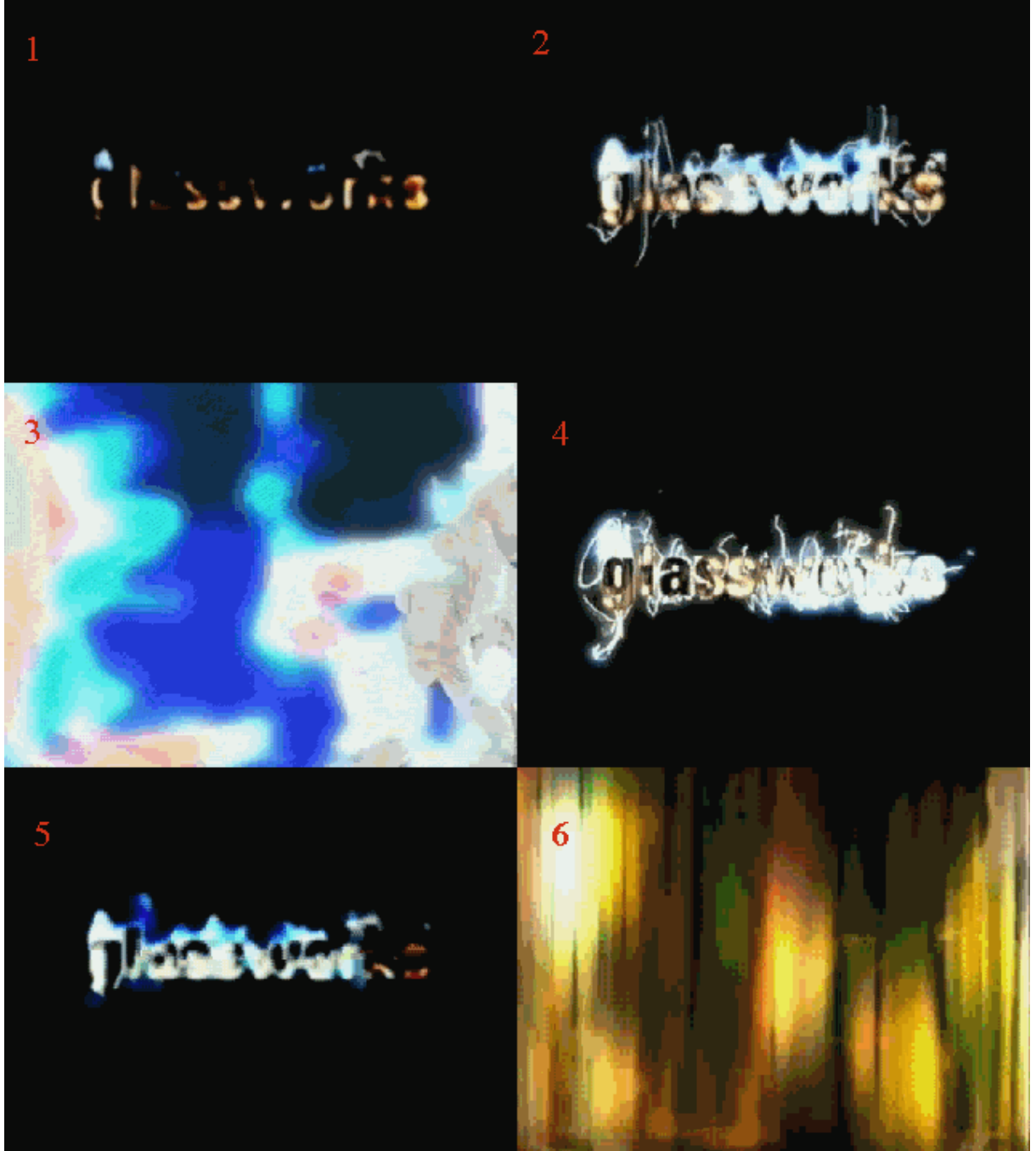
2 647 852 bayt

Çıktı dosyasının boyutu:

506 559 bayt

Sıkıştırma oranı:

**% 80,9**



Parlayıp renk deęiřtiren 'glassworks' yazısını ieren grntde zaman zaman plan harici deęiřik kareler araya giriyor. Grnt toplam 47 kareden oluřuyor ve karelerin her biri 256 renkli 320x240 piksel boyutunda.

bmp dosyalarının toplam boyutu:

3 660 266 bayt

ıktı dosyasının boyutu:

671 533 bayt

Sıkıřtırma oranı:

**% 81,65**

4 karesi aşağıda yer alan 14 kareden oluşan görüntüde her kare 256 renkli ve 320x240 piksel boyutunda.

bmp dosyalarının toplam boyutu:

1 090 292 bayt

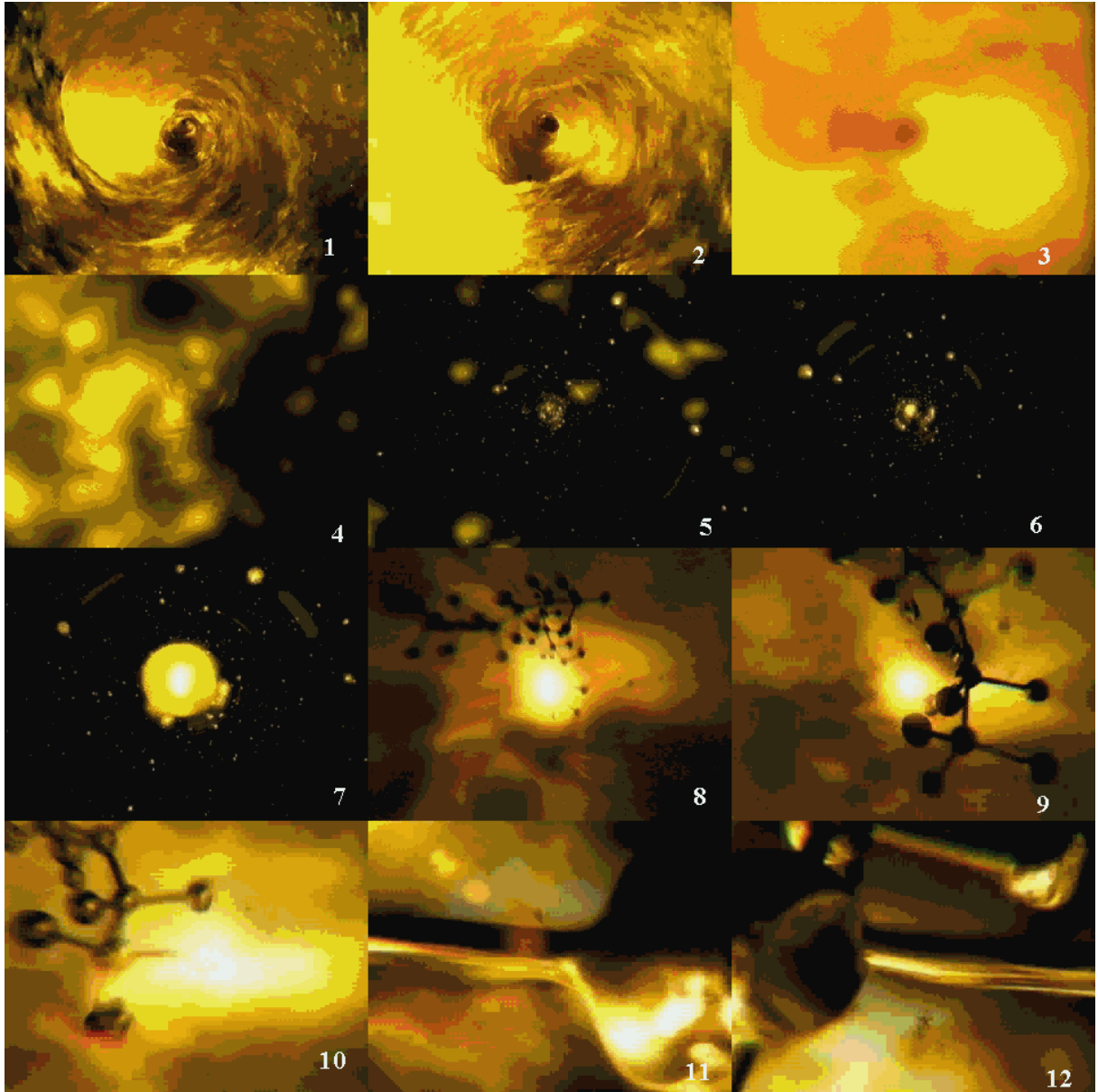
Çıktı dosyasının boyutu:

671 969 bayt

Sıkıştırma oranı:

**% 38,3**





43 kareden oluşan ve her karesi 256 renkli, 320x240 piksel boyutunda olan görüntünün 12 karesi yukarıda.

bmp dosyalarının toplam boyutu:

3 348 754 bayt

Çıktı dosyasının boyutu:

1 675 909 bayt

Sıkıştırma oranı:

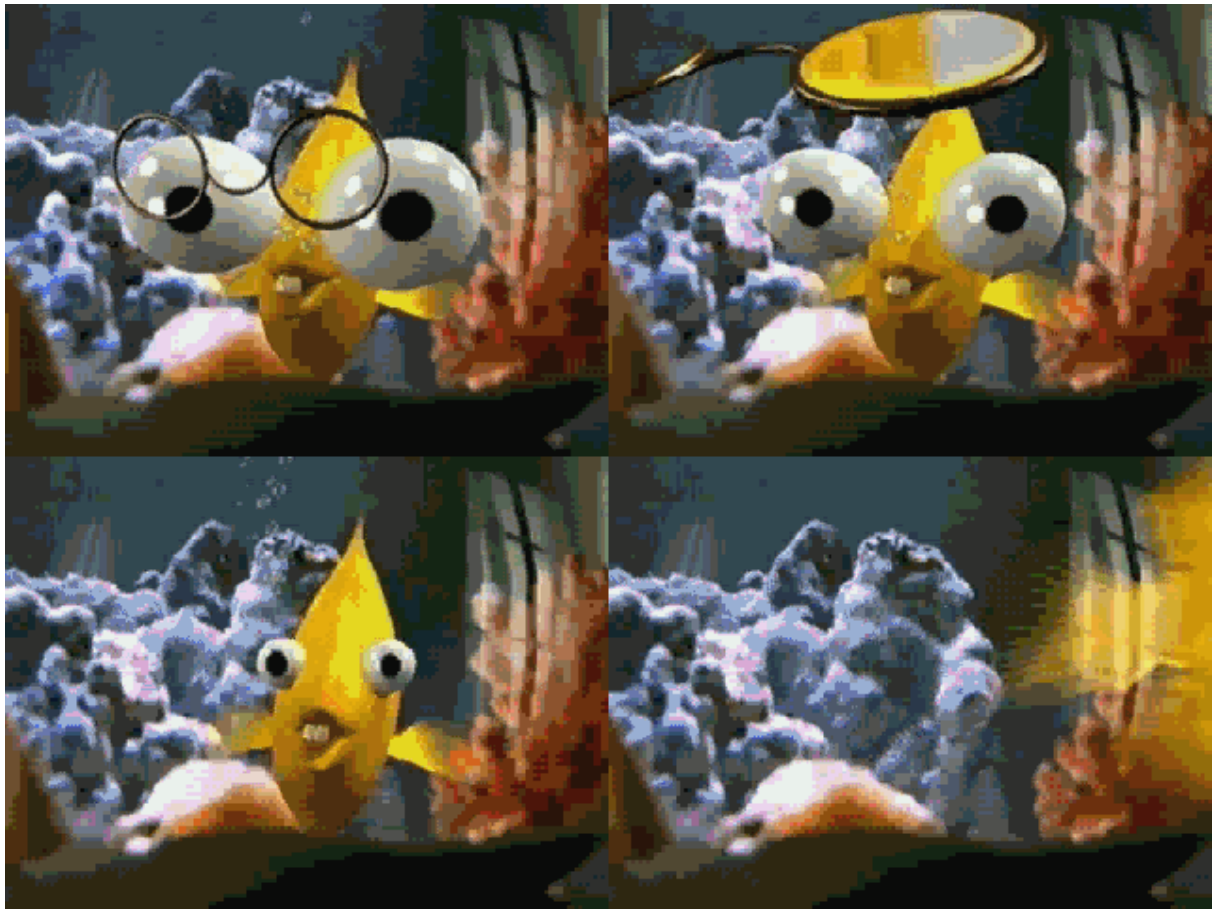
**% 49,95**

256 renkli 320x240 piksel boyutunda 4 karesi aşağıda yer alan ve 21 kareden oluşan görüntü için alınan sonuçlar şöyledir:

bmp dosyalarının toplam boyutu: 1 635 438 bayt

Çıktı dosyasının boyutu: 1 220 347 bayt

Sıkıştırma oranı: % 25,4



Anlatılan algoritmanın uygulanması sonucu ortaya çıkan programın değişik özellikteki video görüntüleri üzerinde denenmesi sonucu %80 'e varan başarı elde edilmiştir. Önceden tahmin edilebileceği gibi hareketin az olduğu görüntülerde daha fazla sıkıştırma sağlanmıştır. Bu çalışma her türlü video görüntüsünü az ya da çok sıkıştıracaktır (en azından görüntü haritalarının tekrar tekrar yazılmamasıyla bir sıkışma sağlanacaktır).

Programın daha çok başarı sağlayabileceği görüntüler olabileceği gibi (örneğin haberleri sunan spiker) başarının düşük olacağı görüntüler de olacaktır. Görüntülerin özellikleri birbirinden çok farklı olduğundan ortalama bir başarıdan söz etmek mümkün değildir.

## E. ARAÇ GEREÇLER

Anlatılan algoritmayı uygulayan ve çıktıyı tekrar bmp dosyalarına geri çeviren programlar MATLAB® *for Windows* programı ile yazılmıştır. Programı test etmek amacıyla .avi dosyalarından software olarak dağıtımı yapılan Gamani GIF Movie Gear programının 3.0 versiyonu ile kareler ayrı ayrı .gif dosyaları olarak kaydedilmiş, ve yine MATLAB 'da yazılan basit bir arayüzle bu dosyalar .bmp formatına çevrilmiştir.

Çalışma aşağıda belirtilen özelliklere sahip kişisel bilgisayar ile gerçekleştirilmiştir.

|                 |                                  |
|-----------------|----------------------------------|
| İşlemci         | : Pentium III 550                |
| RAM             | : 3 x 64 = 192 MB                |
| Sabit Disk      | : 15 GB                          |
| İşletim Sistemi | : Windows 98 İkinci Sürüm Türkçe |

## F. KAYNAKÇA

Math Works Web Site  
<http://www.mathworks.com>

Çeşitli sıkıştırma algoritmaları ve veri yapıları hakkında bilgi:  
<http://www.programmersheaven.com>

Data Compression Reference Center:  
<http://www.rasip.fer.hr/research/compress/basic/index.html>

Veri Sıkıştırma Algoritmaları hakkında genel bilgi:  
<http://www.programlama.com>

Matrix Compression Techniques  
<http://www.angelfire.com/ri/hirschberg/>

Data Compression Library  
<http://www.dogma.net/DataCompression/>

Video Compression  
<http://www.cs.sfu.ca/undergrad/CourseMaterials/CMPT365/material/notes/Chap4/Chap4.2/Chap4.2.html>

Video File Formats  
<http://128.173.40.129/~netinfo/notes/chap1/digitalvideo/videoff.html>

MPEG Compression Algorithm  
<http://icsl.ee.washington.edu/~woobin/papers/General/node2.html>