İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

JPEG XR BAŞARIMI

YÜKSEK LİSANS TEZİ

Sezer BAĞLAN

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Telekomünikasyon Mühendisliği Programı

OCAK 2014

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

JPEG XR BAŞARIMI

YÜKSEK LİSANS TEZİ

Sezer BAĞLAN
(504081328)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Telekomünikasyon Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Melih PAZARCI

OCAK 2014

İTÜ, Fen Bilimleri Enstitüsü'nün 504081328 numaralı Yüksek Lisans Öğrencisi **Sezer BAĞLAN**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **" JPEG XR BAŞARIMI"** başlıklı tezini aşağıdaki imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Prof. Dr. Melih PAZARCI** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Prof. Dr. Bilge GÜNSEL KALYONCU** İstanbul Teknik Üniversitesi	.....
	Prof. Dr. Muhittin GÖKMEN İstanbul Teknik Üniversitesi	.....
		.....

Teslim Tarihi : **16 Aralık 2013**
Savunma Tarihi : **22 Ocak 2014**

Eşime ve aileme,

ÖNSÖZ

Yüksek Lisans eğitimim boyunca hiçbir konuda desteğini esirgemeyen, en zor zamanlarımda beni yüreklendiren ve inanılmaz sabır gösteren danışmanım sayın Prof. Dr. Melih PAZARCI'ya teşekkürlerimi sunarım.

Yüksek Lisans eğitimime başlamamda ve sürdürmemde bana hep destek olan değerli eşim Cemre ATAŞ BAĞLAN'a da teşekkürlerimi sunarım.

Ocak 2014

Sezer BAĞLAN
Elektrik-Elektronik Müh.

İÇİNDEKİLER

Sayfa

KISALTMALAR

AC : Alternative Current
App : Appendix
BMP : Bitmap
bpp : bits per pixel
DC : Direct Current
DCT : Discrete Cosine Transform
EXIF : Exchangeable Image File Format
FCT : Forward Core Transform
GIF : Graphics Interchange Format
HP : High Pass
ICC : International Color Consortium
IEC : International Engineering Consortium
IFF : Interchange File Format
ISO : International Organization for Standardization
ITU-T : International Telecommunications Union Telecommunication
JPEG : Joint Picture Experts Group
JPEG XR : Joint Picture Experts Group Extended Range
LP : Low Pass
MCU : Minimum Coded Unit
MSE : Mean Square Error
OT : Overlap Transform
PSNR : Peak Signal to Noise Ratio
QP : Quantization Parameter
RLE : Run-Length Coding
SSIM : Structural Similarity
TARGA : Truevision Advanced Raster Graphics Adapter
TIFF : Tagged Image File Format
VLC : Variable-Length Coding
XMP : Extensible Metadata Platform

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

JPEG XR BAŞARIMI

ÖZET

Bir görüntü sıkıştırma standardı olarak JPEG XR teknolojisi Microsoft'un geliştirdiği HD Photo standardı üzerine bina edilmiş, bilinen ve kullanılan en yaygın görüntü sıkıştırma standardı olan JPEG'den iki kat daha iyi sıkıştırma oranına sahip, bunun yanında sıkıştırma oranı JPEG'e göre daha iyi olan JPEG-2000'den de işlem karmaşıklığı açısından daha az karmaşık olan ISO/IEC JPEG Standardı Komitesi tarafından organize edilmiş bir standarttır.

Günümüzde birçok çoklu ortam uygulaması yüksek sıkıştırma oranlarına ve yüksek görüntü kalitesine sahip görüntü sıkıştırma teknolojilerine ihtiyaç duymaktadır. Tüketici elektroniğinde yüksek kalite, yüksek sıkıştırma oranları ve düşük işlem maliyetleri önemli faktörler haline gelmektedir. Bu gereksinimleri karşılamak amacıyla ortaya çıkan JPEG XR standardı gelecekte JPEG'in yerini almayı hedeflemektedir.

Bu çalışmada JPEG ve JPEG XR standartları yapısal olarak incelenmiş, bu standartları oluşturan blokların sıkıştırılmış görüntüler üzerindeki etkileri nesnel ve görsel olarak ortaya konulmuştur. Karşılaştırmalar esnasında referans olması açısından JPEG-2000 standardı da kullanılacaktır.

Nesnel karşılaştırma esnasında farklı özelliklere sahip test görüntüleri kullanılmıştır. Bu test görüntüleri JPEG, JPEG-2000 ve JPEG XR kodlandıktan sonra elde edilen kodlanmış görüntüler yine aynı standartlara ait kod çözücüler vasıtasıyla çözülüp aynı formata dönüştürülerek PSNR (Tepe sinyal gürültü oranı) ve SSIM (Yapısal benzerlik) metrikleri hesaplanarak performans karşılaştırması yapılmıştır. Elde edilen sonuçlar ise PSNR ve SSIM değerlerinin sıkıştırma oranlarına göre nasıl değiştiğini gösteren grafikler vasıtasıyla ortaya koyulmuştur.

Görsel karşılaştırmada da farklı özelliklere sahip test görüntüleri kullanılarak benzer sıkıştırma oranlarına sahip JPEG, JPEG-2000 ve JPEG XR kodlanmış görüntülerin görsel olarak karşılaştırmaları yapılmıştır. Bunun yanısıra sıkıştırma esnasında kullanılan parametrelerin görüntüye etkileri de ortaya koyulmuştur.

Bu çalışmada JPEG XR'nin nesnel ve görsel olarak JPEG standardını geride bıraktığı görülmüştür. Bunun yanı sıra JPEG-2000 ile nesnel karşılaştırma sonucu görece yüksek sıkıştırma oranlarında benzer performans gösterdiği, görece düşük sıkıştırma oranlarında da JPEG-2000'den daha iyi performans sergilediği görülmüştür. JPEG-2000 ile görsel karşılaştırmada ise birbirlerine büyük üstünlükler sağlayamadıkları ve görsel performanslarının test görüntülerine göre değiştiği görülmüştür.

JPEG XR
PERFORMANCE

SUMMARY

As an image compression technology, JPEG XR is built on HD Photo standard developed by Microsoft and it is organized by ISO/IEC JPEG Standard Committee. JPEG XR has compression ratios almost two times better than JPEG standard which is a well-known and the most widely used image compression standard. Furthermore JPEG XR has less computational complexity compared to JPEG-2000 standard which has better compression ratios than JPEG standard.

Nowadays many multi-media applications need image compression technologies having high compression ratios and high image quality. High quality, high compression ratios and low costs are becoming important factors in consumer electronics. For high quality of multi-media applications, the extension of color range is also becoming important. In the past, the digital cameras and display devices mostly have 8 bits per channel. Due to improvement of display device technology, JPEG XR is designed for high dynamic range (HDR) and the high definition photos. The XR of JPEG means that JPEG XR supports extended range of information per channel. JPEG XR standard aims to take JPEG's place in order to meet these requirements.

JPEG XR has several improvements over JPEG, including better compression, lossless compression, tile structure support, more color accuracy support, transparency map support. JPEG XR supports higher compression ratios in comparison to JPEG while encoding an image with similar quality levels. JPEG XR supports lossless compression while JPEG does not. A JPEG XR encoded image can be separated into regions called tiles. The data of each tile can be decoded separately which provides flexibility. JPEG is capable of supporting 8-bit integer in gray scale and 24-bit integer per pixel for true color, alternatively JPEG XR is capable of supporting 64-bit integer and 128-bit floating number per pixel. JPEG XR has also alpha channel support to represent transparency.

The structures of JPEG XR and JPEG standards are investigated in this study. The typical encoding process is composed of stages including: color space transformation, downsampling, block splitting, frequency transformation, quantization and finally entropy coding. These stages differ for JPEG and JPEG XR. In color space transformation of JPEG from RGB to YCbCr, a linear transformation is used which results in data loss due to rounding errors. However JPEG XR has no data loss in this stage. In the frequency transformation stage, JPEG uses 8x8 blocks whereas JPEG XR uses 4x4 blocks. So, the frequency transformation of JPEG XR is more fine-grained. In the frequency transformation stage of JPEG, there is a small amount of loss due to DCT rounding erros. However in JPEG XR the frequency transformation is integer based and invertible, so it is lossless. JPEG XR has an adaptive entropy coding

technique in comparison to JPEG's zig-zag method. In this study, the objective and subjective effects of the stages that make up these standards on the compressed images are revealed. In performance comparison stage, JPEG-2000 standard is also used as a reference.

During objective comparison test images having distinctive properties are used: Lenna, Baboon, Barbara, Lighthouse and Ruler. Lenna is classical test image having smooth color changes and borders. Baboon is a color image which is hard to compress with lots of details, great variation of color and a large amount of texture. Barbara is a color image with many stripes having face and skin. Lighthouse is a color image with many details having also natural sky. Ruler is an artificial image with text and lines which is hard to compress. Firstly, test images are encoded via JPEG, JPEG-2000 and JPEG XR standards. Immediately afterwards, encoded images are decoded with the same standards that they are encoded. Finally, the obtained images are transformed to the same image format for performance measurement. PSNR (Peak signal noise ratio) and SSIM (Structural similarity) metrics are used for performance measurement and comparison. PSNR is an engineering metric for the ratio between possible power of a signal and the power of corrupting noise. It is usually expressed in terms of logarithmic decibel scale. PSNR is an approximation to human perception of reconstruction quality. PSNR metric is only valid when it is used to compare results from the same codec and same content. SSIM index is a metric for measuring similarity between two images. It is a full reference metric which means that measuring image quality based on an uncompressed image as a reference. SSIM is designed to improve consistency with human eye perception. The results obtained during objective comparison are revealed by means of graphs showing the variation of PSNR and SSIM metrics according to the compression ratios in bits per pixel.

During subjective -visual- comparison JPEG, JPEG-2000 and JPEG XR encoded test images having similar compression ratios are used. For subjective comparison, the compression ratio is chosen as 100. Since JASPER codec supports compression with rate parameter, test images are compressed with 0.01 rate with JASPER codec firstly. After JPEG-2000 compression, the test images are encoded with JPEG and JPEG XR to obtain equivalent or nearest file sizes (compression ratios). Finally the encoded images having similar compression ratios are compared visually. Besides, the effects of parameters used during the compression on the images are also revealed.

In this study it is exhibited that JPEG XR outperforms JPEG both in objective and subjective -visual- comparisons. On the other hand in the objective comparison of JPEG XR and JPEG-2000, it is revealed that in relatively high compression ratios JPEG XR and JPEG-2000 show similar performances, however in relatively low compression ratios JPEG XR has better performance than JPEG-2000.

In subjective comparison of JPEG XR and JPEG-2000, it is obvious that they do not outperform each other distinctively and their visual performances depend on the used test images. The visual performance of JPEG XR does not only depend on the quality parameter but also the overlap transform parameter. When the overlap transform parameter "l" is 2, the visual performance of JPEG XR is almost similar to JPEG 2000. When "l" is 1, it's probable to experience some blocking artifacts on the image.

Google also aims to take JPEG's place with its new picture format WebP. In this study, a comparison of Google's WebP and JPEG XR is also revealed from reference studies in literature. Since the spread of JPEG-2000 usage is limited and JPEG will not be able to meet the increasing need of display technologies in near future, JPEG XR can take JPEG's place for the image compression need. It will be beneficial and advantageous to use JPEG XR in soft and hard platforms due to its low computational complexity and capability of compressing high definition photos.

1. GİRİŞ

Geride bıraktığımız yıllarda gerçekleşen büyük gelişmeler günümüzde görüntü sıkıştırmayı çok önemli hale getirmiştir. Bunlardan ilki sayısal (dijital) fotoğrafa geçilmesi, diğeri ise hayatımızın vazgeçilmezi haline gelen internettir. İnternetin yaygınlaşıp mobil hale gelmesi, dijital fotoğrafçılığın mobil aygıtlarla yapılabilir olması, çok fazla görüntünün hem kaydedilmesine hem de internet ortamında paylaşım ve transferine olanak sağlamıştır. İnsanlar neredeyse her anlarını dijital olarak fotoğraflayıp ya kaydetmekte ya da internet üzerinden paylaşmaktadır.

ISO/IEC JPEG standardı [5] bilinen en yaygın görüntü sıkıştırma standardı konumundadır. İnternet üzerinden indirdiğimiz ya da yüklediğimiz görüntülerin çoğunun JPEG formatında olduğu çok açıktır. Fakat gelişen teknoloji beraberinde daha yüksek hız, daha az yer ihtiyacı, daha yüksek çözünürlük ve daha yüksek kalite getirmeyi amaçladığı için JPEG standardı şu an için hala popüler olsa bile yakın gelecekte varlığını sürdürmekte zorlanabilir.

JPEG standardından sonra ortaya çıkan diğer bir standart ise JPEG-2000 [6] standardıdır. JPEG-2000 digital sinema uygulamaları, uzaktan algılama ve tıbbi görüntüleme gibi münferit alanlarda başarı sağlamıştır. Sıkıştırma oranı olarak JPEG standardıyla karşılaştırıldığında önemli bir sıkıştırma oranına sahip olsa da internet ortamında kullanımı pek yaygınlaşamamıştır.

JPEG-2000'den sonra yeni standart JPEG XR [7] ortaya atılmıştır. Bu yeni standart hem JPEG'in bazı kısıtlarını ortadan kaldırırken hem de sıkıştırma oranı olarak JPEG-2000'e bir alternatif olarak sunulmuştur. JPEG-2000'e göre daha az işlem karmaşıklığına sahip bir algoritma olmasından dolayı mobil uygulamalarda ve daha büyük boyutlu görüntülerin sıkıştırılmasında kullanılması düşünülmüştür [8]. JPEG XR temelde yüksek kalite ve verimli bir sıkıştırma oranı sağlayarak, kodlayıcı ve kod çözücü taraflarındaki işlem karmaşıklığını da azaltmayı hedeflemiştir.

Bu çalışmadaki amacımız, JPEG XR standardını incelemek ve JPEG ile JPEG XR'ı karşılaştırmaktır. JPEG-2000 kodlama yapısal farklılığından ötürü bu çalışmada incelenmemiş, fakat bir performans ölçütü olması açısından performans karşılaştırmalarında kullanılmıştır.

Bölüm 2'de JPEG görüntü sıkıştırma standardı hakkında genel bilgi verilecek, 3. bölümde ise JPEG XR görüntü sıkıştırma standardı incelenecektir. 4. bölümde ise bu standartlar arasındaki farklar deneysel olarak ortaya konulacak ve sonuçlar karşılaştırılacaktır.

2. JPEG

2.1 JPEG Hakkında

JPEG'in görüntü sıkıştırma formatı olarak ortaya çıkması, kabul edilebilir kaliteye sahip ve hafızada daha az yer kaplayacak bir standarda olan ihtiyaçtan doğmuştur. Bu ihtiyacı karşılamak isteyen ISO / ITU mühendisleri uluslararası bir araştırma ekibi oluşturarak Joint Picture Expert Group (Birleşik Fotoğraf Uzmanları Grubu) adlı grubu oluşturmuş ve 1992 yılında da JPEG görüntü sıkıştırma standardını ortaya atmışlardır.

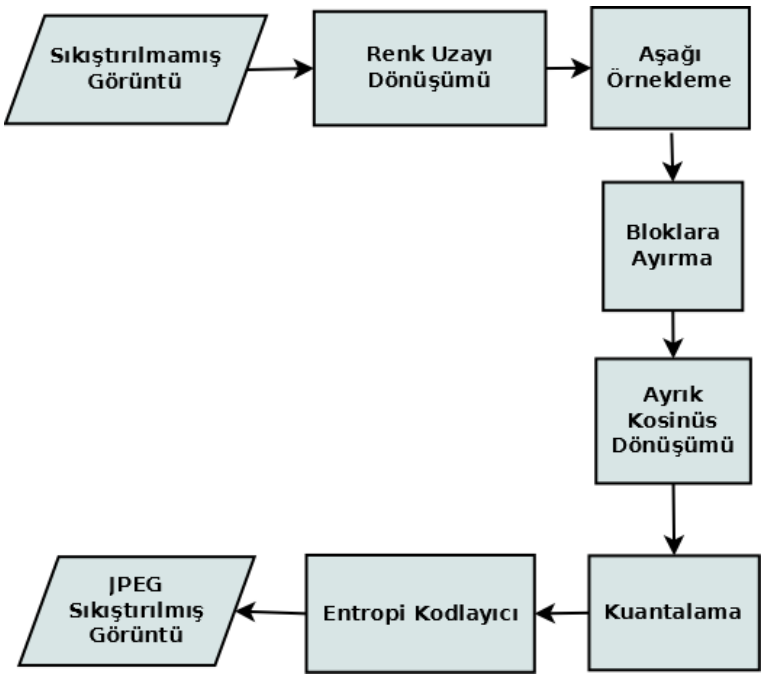
JPEG'den önce kullanılan bazı görüntü formatlarını BMP, TIFF, TARGA, IFF ve diğer bazı görüntü formatları olarak sıralayabiliriz. Bu görüntü formatlarının adı kayıpsız sıkıştırma ile anılmakla birlikte, kayıpsız sıkıştırmanın saylayabildiği kısıtlı sıkıştırma oranları ile sınırlıdırlar. Sırasıyla inceleyecek olursak BMP hiç sıkıştırma uygulamayarak, piksellerin renklerine karşılık gelen 8, 16 veya 24 bit sözcük dizilerinden meydana gelmektedir. TIFF formatı RLE veya Huffman gibi entropik teknikler uygulayarak görüntü boyutunu azaltır. Fakat kullanılan bu teknikler yapıları gereği aynı renk değerliğine sahip çok fazla pikselin olduğu durumlarda işe yaramaktadır. Yani fotoğrafik görüntülerde sıkıştırma işleminde başarı sağlayamamaktadır. GIF formatına baktığımızda 256 renk ile sınırlı olduğundan, bu format da fotoğrafik görüntülerde kullanılmaya pek uygun gözükmemektedir. Sıkıştırma oranlarına bakıldığında ise bu formatların 2 kat sıkıştırma oranından daha büyük oranlara çıkamadığı gözlemlenmiştir.

JPEG hem fotoğrafik görüntülerde sağladığı yüksek sıkıştırma oranları ile hem de İnternetin yaygınlaşması ile görüntü iletiminin önemli hale gelmesinden dolayı ön plana çıkmış ve günümüzde en popüler görüntü sıkıştırma formatı haline gelmiştir.

2.2 JPEG Kodlama

JPEG, görüntüler üzerinde kodlama esnasında belirli işlem bloklarında kullanıcıya esneklik sunan bir standarttır. Yani JPEG kodlama yapılırken istenildiği takdirde bazı

işlemler atlanabilmektedir. Şekil 2.1'de JPEG kodlayıcı sistemin genel yapısı kabaca verilmiştir. Sırasıyla renk uzayı dönüşümü, aşağı örnekleme, bloklara ayırma, ayrık kosinüs dönüşümü, kuantalama ve entropi kodlayıcı bloklarını inceleyeceğiz.



Şekil 2.1: JPEG kodlayıcı sistemin genel yapısı.

2.2.1 Renk uzayı dönüşümü

Öncelikle görüntü RGB renk uzayından diğer bir renk uzayı YCbCr'a dönüştürülmelidir. Y bir pikselin parlaklığını, Cb ve Cr ise renk bileşenlerini oluşturur. YCbCr renk uzayına dönüşüm yapmak algılanabilir kalite seviyesinde büyük etkiler yaratmadan daha büyük sıkıştırma oranları elde etmede kolaylık sağlamaktadır. İnsanın görmedeki kalite algısında parlaklık daha büyük bir etkiye sahiptir. Böylece renk ve parlaklık bileşenleri ayrılıp bu şekilde daha büyük sıkıştırma oranları elde edilebilmektedir. Burada dikkat edilmesi gereken diğer bir nokta, bazı kaynaklarda YCbCr yerine YUV terimi kullanılmasıdır. Bu iki terim zaman zaman birbirilerinin yerine kullanılabilmektedir. Aslında YUV terimi televizyon sistemlerinde analog renk bilgisi kodlamada kullanılmaktadır. Renk bilgisinin dijital kodlanmasında ise YCbCr terimi tercih edilmektedir. Bu yüzden bu çalışma boyunca YCbCr terimi kullanılacaktır.

2.2.2 Aşağı örnekleme

İnsan gözündeki parlaklık ve renk reseptörlerinin yoğunluğuna bağlı olarak insanlar renk bileşenlerine (Cb ve Cr) nazaran bir görüntünün parlaklığında (Y) daha fazla detay görebilirler. Kodlayıcı tarafında kodlama işlemini gerçekleştirirken bu bilgi kullanılarak sıkıştırmada verimlilik artışı sağlanabilmektedir. Örneğin JPEG olarak kodlanmış bir bit dizisinde her pikselin parlaklık değerinin (Y) tutulduğunu, buna karşılık renk bileşenlerinin (Cb + Cr) ise her piksel grubu (2, 4 veya 8 piksellik grup) için bir adet tutulduğunu varsayalım. Bu durumda renk bileşenlerinin getirdiği yük normalde getireceği yükten 2,4 veya 8 kat daha az olacaktır. Çünkü normalde Y, Cb ve Cr aynı miktarda bit sayısı ile tutulmaktadırlar [9].

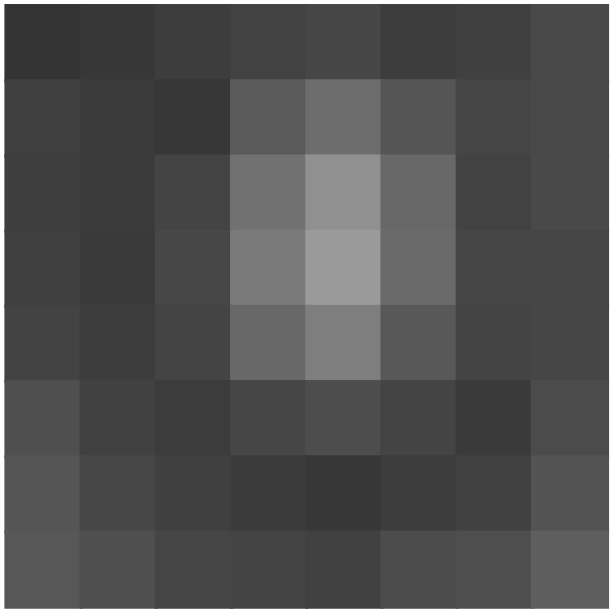
JPEG kodlanmış görüntülerde genellikle kullanılan aşağı örnekleme oranları şu şekilde sıralalanbilir: 4:4:4 (aşağı örneklemenin olmadığı durum), 4:2:2 (2 çarpanı ile yatay yönde azaltılmış aşağı örnekleme), 4:2:0 (2 çarpanı ile hem yatay hem düşey yönde azaltılmış örnekleme).

2.2.3 Bloklara ayırma

Bloklara ayırma aşamasında her kanal (Y, Cb, Cr) 8x8 piksellik bloklara ayrılır ve Şekil 2.2'de görüldüğü gibi bu bloklara MCU (Minimum Kodlanmış Birim, Minimum Coded Unit) denir. Aşağı örnekleme oranlarına göre MCU boyutları değişebilmektedir: 4:4:4 (8x8 piksellik blok), 4:2:2 (16x8 piksellik blok), 4:2:0 (16x16 piksellik blok). Görüntü bilgisinin ya da kanaldaki piksellerin MCU'ların katı olmadığı durumlarda, kalan kısımlar siyah piksellerle ya da sınır pikselleri tekrarlanarak doldurulabilir. Bunların yerine daha gelişmiş boşluk doldurma algoritmaları da kullanılabilir.

2.2.4 Ayrık kosinüs dönüşümü (DCT)

DCT (Ayrık Kosinüs Dönüşümü, Discrete Cosine Transform) bir işareti uzamsal gösterimden frekans gösterimine çevirmede kullanılır. Bir görüntüde, enerjinin büyük bir kısmı düşük frekanslarda toplanmıştır. Bu görüntü frekans bileşenlerine



Şekil 2.2: Gri ölçekte 8x8 MCU.

ayrılıp yüksek frekanslı bileşenleri atılırsa bu görüntü daha az bilgi ile kalitesi çok değişmeden tekrardan tanımlanabilir. JPEG kodlayıcıda renk bileşenlerinden (Y, Cb, Cr) oluşan her 8x8 blok, normalize edilmiş 2 boyutlu DCT ile frekans uzayına dönüştürülür [10]. Bu dönüşümden önce 8x8 blok için her bir pikselin gri seviyesi, 2^n görüntüdeki maksimum gri seviyesini belirtmek üzere piksel değerlerinden 2^{n-1} çıkartılarak tekrardan hesaplanır. Örneğin 8-bitlik bir görüntüde, 8x8 bloğun her bir pikselinden 2^7 çıkarılır. Bu sayede piksel değerleri 0 etrafında toplanmış olur ve bu da DCT sırasında kolaylık sağlamaktadır.

S uzamsal bir görüntüyü ve S_{yx} ise (x,y) kordinatındaki pikseli temsil etsin. S görüntüsünün frekans uzayındaki F görüntüsüne dönüşümü aşağıdaki şekilde tanımlanmaktadır (**2.1**).

$$\begin{aligned} C_u &= \begin{cases} \frac{1}{\sqrt{2}} & u=0 \\ 1 & u\neq 0 \end{cases} \\ C_v &= \begin{cases} \frac{1}{\sqrt{2}} & v=0 \\ 1 & v\neq 0 \end{cases} \\ F_{vu} &= \frac{1}{4}C_vC_u\sum_{y=0}^{N-1}\sum_{x=0}^{N-1}S_{yx}cos\left(v\pi\frac{2y+1}{2N}\right)cos\left(u\pi\frac{2x+1}{2N}\right) \end{aligned} \qquad (\mathbf{2.1})$$

DCT sonrası 8x8 matris incelendiğinde en sol-üst köşede en büyük değerlikli katsayı bulunur. Bu katsayıya DC katsayısı adı verilir. Geriye kalan 63 adet katsayı ise AC katsayılarıdır. DCT'nin burada kullanım amacı görüntü işaretinin büyük bir kısmını bir köşede toplayabilmektir. Bir sonraki adım olan kuantalamada DCT'nin önemi daha belirgin bir biçimde anlaşılacaktır.

2.2.5 Kuantalama

Kuantalama basamağında insan için daha az önemli görsel bilgiyi taşıyan katsayıların (yüksek frekanslı bileşenler) yok edilmesi amaçlanmıştır. Bu, bir önceki DCT basamağında elde edilen 8x8 matrisin bir kuantalama tablosu yardımı ile değerlerinin tekrardan hesaplanması ve sonucun en yakın tam sayıya yuvarlanması ile elde edilir. Yuvarlama işlemi JPEG kodlamada kayıplı sıkıştırmaya neden olan işlemdir. F kuantalanmamış DCT katsayılarını, Q kuantalama matrisini ve B kuantalanmış DCT katsayılarını göstermek üzere, kuantalanmış DCT katsayıları aşağıdaki şekilde tanımlanmaktadır (**2.2**).

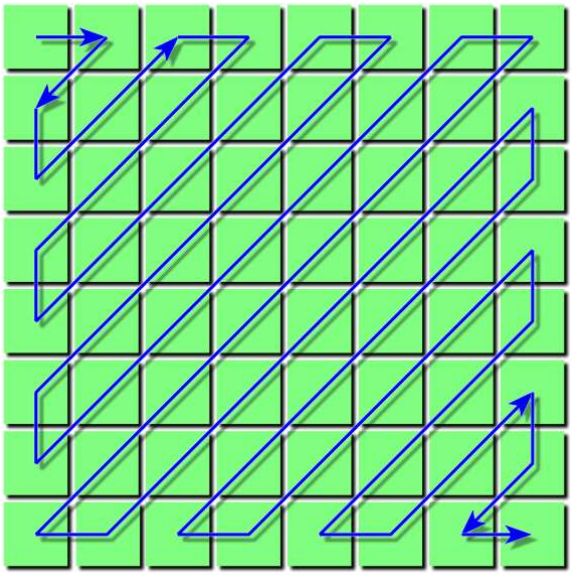
$$B_{j,k} = \text{yuvarla}(\frac{F_{j,k}}{Q_{j,k}}) \quad \text{j = 0, 1, 2,..., 7 ve k = 0, 1, 2,..., 7 için} \qquad (\mathbf{2.2})$$

Kuantalama işleminin sonucunda en düşük değerlikli DCT katsayıları (yüksek frekanslı) 0'a yuvarlanır. Sonuç olarak sol-üst köşede kümelenmiş değerliği 0'dan farklı katsayılara sahip ve geri kalan katsayıların ise 0 olduğu bir matris elde edilir. JPEG kodlama esnasında seçilen sıkıştırma oranları aslında kuantalama basamağı ile birebir ilişkilidir. Sıkıştırma oranı ne kadar artırılırsa 0'a yuvarlanan katsayılar o kadar artacak ve sıkıştırılmış görüntüde kalite azalacaktır.

2.2.6 Entropi kodlayıcı

JPEG kodlamadaki son basamak entropi kodlayıcı basamağıdır. Önceki basamaklarda önemsiz görsel bilgi azaltılmıştı. Entropi kodlayıcının görevi ise entropik teknikler kullanarak sıkıştırılmış görüntünün hafızada daha az yer kaplamasını sağlamaktır. Bu basamakta ilk olarak Şekil 2.3'te de görüldüğü gibi "zigzag" sıralama adı verilen bir yol üzerinden görüntü bileşenleri bir sıraya dizilir. Sıralama bittikten sonra seri-uzunluklu-kodlama (run-length-coding, RLE) adı verilen basit bir sıkıştırma

tekniği uygulanır. Zigzag dizilimin ardından elde edilen 1x64 vektör çok sayıda 0 içerir. RLE'de bu vektör (s,d) çiftleri ile gösterilir. Burada s ardışık sıfırların sayısını, d ise bu 0'lar bittikten sonra gelen ilk 0'dan farklı katsayının değerini gösterir. $(0,0)$ çifti ise 1x64 vektörün sonlandığını gösteren çifttir. Entropi kodlayıcının son adımı ise Huffman sıkıştırmasıdır. Huffman sıkıştırması esnasında veri sözcüklere (bit dizileri) bölünür ve her sözcüğün bu veri içerisindeki bulunma sıklığına göre bu sözcüklere değişken uzunluklu kodlar tanımlanır. Çok sık kullanılan sözcükler için kısa kodlar, bunun yanında çok seyrek kullanılan sözcükler için ise uzun kodlar kullanılır.



Şekil 2.3: JPEG görüntü bileşenlerinin zigzag sıralanması.

JPEG standardı Huffman kodlamanın yanında aritmetik kodlamayı (arithmetic coding) da desteklemektedir. Aritmetik kodlama matematiksel olarak Huffman kodlamadan üstün olsa da hem kodlayıcı ve kod çözücü tarafta daha yavaş çalışması sebebiyle daha az kullanılmaktadır.

2.3 JPEG Kod Çözücü

JPEG standardı yapısı itibarıyla kodlayıcı ve kod çözücü tarafta simetriktir. JPEG kodlama ile sıkıştırılmış bir görüntü sıkıştırma için yapılan tüm işlemler tersten uygulanarak çözülebilir. JPEG kodlanmış görüntüye önce Huffman kod çözücü uygulanır. DC ve AC katsayılara sahip 8x8 blok elde edilir. Katsayılar ters kuantalama tablosu kullanılarak yeniden hesaplanır. Elde edilen 8x8 matrise ters DCT uygulanır

ve kodu çözülen dosyanın sıkıştırma esnasında seçilen ayarlarına göre YCbCr-RGB dönüşümü yapılır.

3. JPEG XR

3.1 JPEG XR Hakkında

JPEG XR, teknolojisi Microsoft tarafından geliştirilmiş HD Photo üzerine inşa edilmiş, ITU-T ve ISO/IEC tarafından standartlaştırılmış yeni bir görüntü sıkıştırma standardı olarak karşımıza çıkmaktadır. JPEG XR'nin, JPEG standardıyla karşılaştırıldığında 2 kat sıkıştırma performansı sağlaması ve JPEG-2000 ile karşılaştırıldığında düşük işlem karmaşıklığına sahip olmasından ötürü gelecekte yaygın bir standart haline gelmesi beklenmektedir [11]. JPEG XR standardı 5 bölümden oluşan "Information Technology - JPEG XR Image Coding System" (ISO / IEC 29199) dökümanında tanımlanmıştır:

1. Bölüm: Sistem Mimarisi (ISO / IEC 29199-1): Spesifikasyonlar, kodlayıcı ve kod çözücü tasarımının anlatıldığı bölümdür [7].

2. Bölüm: Görüntü Kodlama Spesifikasyonu (ISO / IEC 29199-2): Kodlama formatının detaylı bir biçimde anlatıldığı bölümdür [2].

3. Bölüm: Hareketli JPEG XR (ISO / IEC 29199-3): Hareketli görüntü dizilerinin depolanması için gerekli kodlamayı anlatan bölümdür [12].

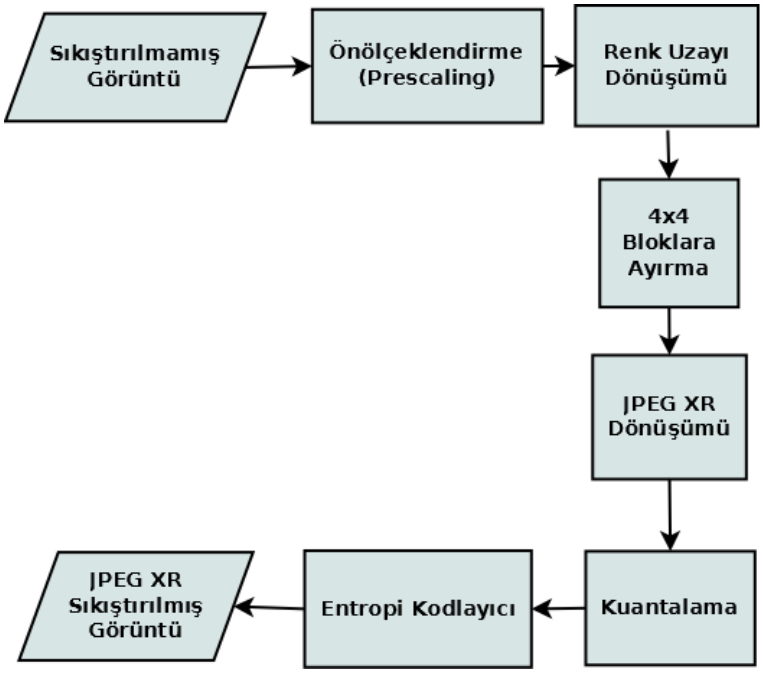
4. Bölüm: Uygunluk Testleri (ISO / IEC 29199-4): Bölüm 2'de anlatılan spesifikasyonları test etmek için bir takım testler gösterir [13].

5. Bölüm: Referans Yazılım (ISO / IEC 29199-5): Bölüm 2 için gerekli referans yazılımdır. Uygunluk testleri için yeni kodlayıcı ve kod çözücülerin oluşturulmasında kullanılır [14].

3.2 JPEG XR Kodlama

JPEG XR görüntü sıkıştırma algoritması kabaca renk uzayı dönüşümü, dönüşüm kodlaması, kuantalama ve entropi kodlama basamaklarından oluşmaktadır. Kodlama algoritmasının genel yapısı Şekil 3.1'de verilmiştir. Sıkıştırılmamış görüntü öncelikle birbirini kesmeyen dikdörtgen bloklara ayrılır ve RGB renk uzayından YUV renk uzayına dönüştürülür. Her renk kanalı (Y, U, V) için ayrı ayrı birer renk düzlemi

oluşturulur ve bu renk düzlemleri 4x4 bloklara ayrılır. Bloklara ayırma basamağından sonra 2 boyutlu FCT (Forward Core Transform) uygulanır. Bir sonraki basamak olan kuantalama basamağı daha az önemli bilginin yok edildiği basamaktır. En son basamak ise kalan önemli bilginin depolanması veya iletimi sırasında daha az veri kullanılmasını sağlayan entropi kodlama basamağıdır.



Şekil 3.1: JPEG XR kodlayıcı sistemin genel yapısı.

3.2.1 Ön ölçeklendirme (Prescaling)

Ön ölçeklendirme basamakları 16-bit işaretsiz tamsayı (BD16), 16-bit işaretli tamsayı (BD16S), 32-bit işaretsiz tamsayı (BD32) ve 32-bit işaretli tamsayı (BD32S) için opsiyoneldir. Ön ölçeklendirme genellikle giriş verisinin 27/24 bit uzunluğundan daha büyük olduğu durumlarda kullanılır. Örneğin 16-bit veri bulundurulan durumlarda genellikle ön ölçeklendirme ihmal edilir.

Kodlayıcı tarafta giriş verisi sağa doğru *m* bit kaydırılarak verinin 27/24 bit veya daha az bit ile ifade edilmesi sağlanır. Veri ölçeklendirildiğinde 27-bit limiti uygulanırken, verinin ölçeklendirilmediği durumlarda 24-bit limiti uygulanır. Sağa kaydırma işlemi sırasında kaydırılan *m* bit sayısı ise sıkıştırılmış görüntünün başlık kısmında ilgili alanda tutulmaktadır.

3.2.2 Renk uzayı dönüşümü

Kodlayıcı tarafta harici RGB formatlarından dahili YUV formatlarına renk dönüşümü yapılır. Kod çözücü tarafta ise bu işlemin tersi uygulanır. RGB formatından YUV444 formatına dönüşüm aşağıdaki şekilde tanımlanmaktadır (**3.1**).

$$\begin{aligned} V &= B - R \\ t &= R - G + \lceil \frac{V}{2} \rceil \\ Y &= G + \lfloor \frac{t}{2} \rfloor \\ U &= -t \end{aligned} \tag{3.1}$$

Harici CMYK formatından dahili YUVK formatına ters dönüşüm de aşağıdaki şekilde tanımlanmaktadır (**3.2**).

$$\begin{aligned} V &= c - y \\ U &= c - m - \lfloor \frac{V}{2} \rfloor \\ Y &= k - m - \lfloor \frac{U}{2} \rfloor \\ K &= k - \lfloor \frac{Y}{2} \rfloor \end{aligned} \tag{3.2}$$

RGB formatından YUV 4:2:X formatlarından birine dönüşüm yapmak için aşağı örnekleme yapılmalıdır. Renksiz (monochrome) dönüşümlerde YUV 4:4:4 dönüşümü uygulanarak kodlayıcı tarafta U ve V bileşenleri silinebilir. Renk dönüşümü işlemi tamamen terslenebilir olup renk döşümü esnasında bilgi kaybı yaşanmamaktadır.

3.2.3 Bloklara ayırma

Bloklara ayırma işlemi açıklanmadan önce JPEG XR görüntü sıkıştırma algoritmasındaki hiyerarşik yapıyı gözden geçirmek faydalı olacaktır. Bu hiyerarşik yapı 5 katmandan oluşur:

1. Piksel, belirli bir renk kanalındaki belirli bir noktanın tam sayı değerliğidir.

2. Blok, aynı renk kanalındaki komşu 4x4 piksellerin oluşturduğu matristir.

3. Makro-blok, hem parlaklık hem de renk bileşenlerini içeren komşu 4x4 blokların oluşturduğu matristir.

4. Karo (Tile), görüntünün belirli bir bölümünü ifade eden ve makro-blokların

oluşturduğu yapıdır.

5. Görüntü.

Blok tanımından da anlaşılabileceği gibi JPEG XR algoritmasında görüntü 4x4 piksellik bloklara ayrılmaktadır. Görüntüler her zaman 16'nın katı olacak şekilde boyutlara sahip olmayabilirler. Bir görüntünün eni veya boyu 16'nın bir katı değilse görüntü sağa ve aşağı yönde 16'nın en yakın katına ulaşacak şekilde genişletilir. Bu işlem genelde sınır piksellerin yatay ve düşey yönde tekarlanarak 16'nın katı boyutlarda bir görüntü elde edilmesiyle yapılır.

3.2.4 JPEG XR dönüşümü

JPEG XR kodlama algoritması [2] iki katmanlı hiyerarşik bir dönüşüm [7] üzerine inşa edilmiştir. Bu dönüşümlerden ilki FCT (Forward Core Transform), diğeri ise OT (Overlap Transform) dönüşümleridir.

3.2.4.1 FCT (Forward Core Transform)

Dönüşümün gerçeklenebilmesi için görüntü her renk kanalı için 4x4 makro-bloklara bölünür. Her bir makro-blok önceden de bahsedildiği gibi 16 adet 4x4 piksellik birbirini kesmeyen bloklardan oluşmaktadır. Bu blokların herbirine FCT dönüşümü uygulanır ve çıktı olarak birinci basamak için 1 DC ve 15 AC bileşen elde edilir. Her bloktaki 1 adet DC bileşeniyle tekrardan bir 4x4 blok oluşturulur ve tekrar bir FCT dönüşüm uygulanır. Bu işlem sonucunda da ikinci basamak için yine 1 DC ve 15 AC bileşen elde edilmiş olur. En son olarak bu ikinci basamakta elde edilen bileşenler makro-bloğu oluşturan her bloğun ilk pikseline yerleştirilir. Şekil 3.2'de FCT dönüşümünün ikinci basamağı sonrasında bir makro-bloğa ilk basamakta elde edilen 240HP (Yüksek Geçiş) ve ikinci basamakta elde edilmiş olan 15LP (Alçak Geçiş) ve 1 DC katsayıların nasıl yerleştirildiği gösterilmektedir.

FCT dönüşümü 3 temel 2x2 filtreleme işleminden oluşur:

1. 2x2 Hadamard Dönüşümü: T2x2h.
2. 1 Boyutlu Çevrim: TOdd.
3. 2 Boyutlu Çevrim: TOddOdd.

Şekil 3.2: JPEG XR makro-blok katsayı yerleşimi.

FCT dönüşümü Şekil 3.3'te de görüldüğü gibi her bir 4x4 piksellik bloğa iki farklı basamakta uygulanır. Her basamakta 4 adet 2x2 dönüşüm simultane bir şekilde ya da rastlantısal bir sıralamayla uygulanır. İkinci basamağa ait dönüşümler sadece ilk basamak bittiği takdirde uygulanabilir.

İkinci dereceden Hadamard matrisi şu şekilde tanımlanır (**3.3**):

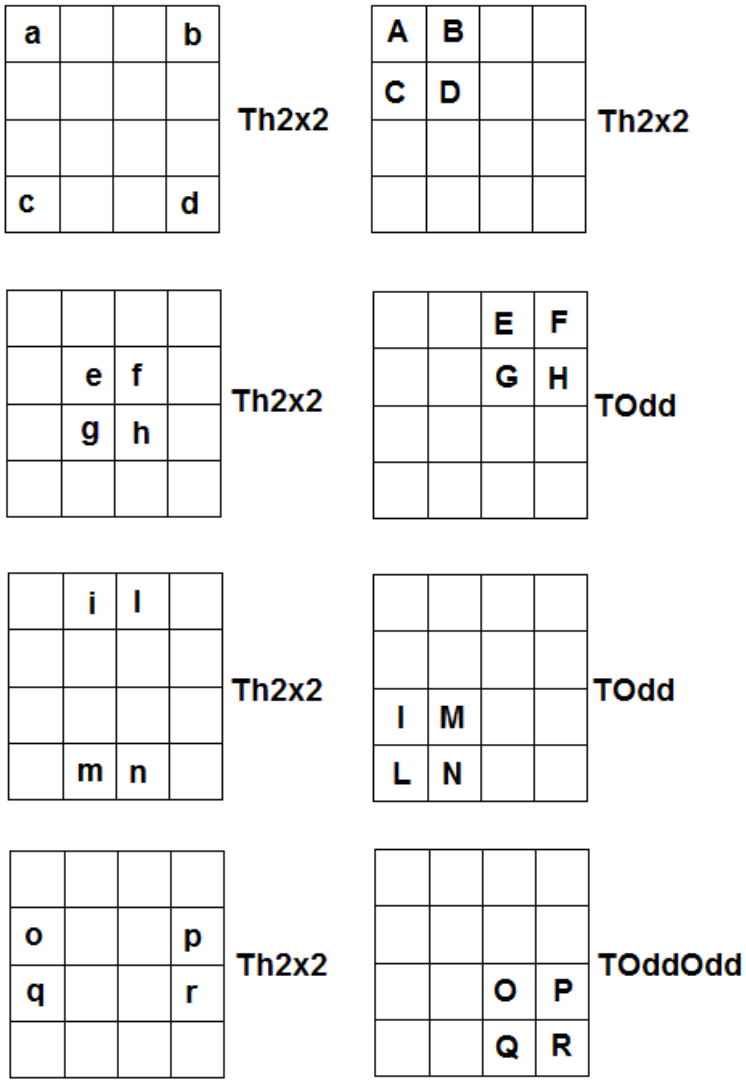
$$H_{2,2} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.3}$$

2j. dereceden Hadamard matrisi ise şu şekilde tanımlanır (**3.4**):

$$H_{2j,2j} = \begin{bmatrix} H_{j,j} & H_{j,j} \\ H_{j,j} & -H_{j,j} \end{bmatrix} \tag{3.4}$$

Ters Hadamard matrisleri işe şu şekilde hesaplanır (**3.5**):

$$H_{j,j}^{-1} = \frac{1}{j} H_{j,j} \tag{3.5}$$



Şekil 3.3: FCT dönüşümü basamakları [1].

f görüntüyü, F ise dönüştürülmüş görüntüyü göstermek üzere Hadamard Dönüşümü ve tersi şu şekilde tanımlanır (**3.6**):

$$\begin{aligned} F &= H_{M,M} f H_{N,N} \\ f &= \frac{1}{MN} H_{M,M} F H_{N,N} \end{aligned} \tag{3.6}$$

2x2 Hadamard dönüşümü için kod taslağı Çizelge 3.1'de gösterilmiştir. Burada a, b, c, d orijinal değerlikleri gösterirken A, B, C, D ise dönüşüm sonrası oluşan değerlikleri göstermektedir. R ise sadece 0 veya 1 değeri alabilen bir faktördür.

Çizelge 3.1: 2x2 Hadamard dönüşümü kod taslağı [1].

Th2x2 kod taslağı
e = a + d
f = b - c
T1 = (e - f + R)/2
T2 = c
C = T2 - d
D = T1 - T2
A = e - D
B = f + C

TOdd ve TOddOdd işlemleri için gerekli 2-nokta çevrim (rotasyon) operatörü şu şekilde tanımlanır (**3.7**):

$$T_R = \begin{bmatrix} -\cos(\frac{\pi}{8}) & \sin(\frac{\pi}{8}) \\ \sin(\frac{\pi}{8}) & \cos(\frac{\pi}{8}) \end{bmatrix} \tag{3.7}$$

Kronecker çarpımı [2] istenilen boyuttaki iki matrisin bir blok matris oluşturması için kullanılan bir işlem olarak tanımlanabilir. A matrisi $(m \times n)$, B matrisi ise $(p \times q)$ boyutlarında birer matris olsun. Bu iki matrisin Kronecker çarpımı şu şekilde tanımlanır (**3.8**):

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}, (m \times p)(n \times q) \text{ bir matris döndürür.} \tag{3.8}$$

daha açık bir biçimde (**3.9**):

$$A \otimes B = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \cdots & a_{11}b_{1q} & \cdots & \cdots & a_{1n}b_{11} & a_{1n}b_{12} & \cdots & a_{1n}b_{1q} \\ a_{11}b_{21} & a_{11}b_{22} & \cdots & a_{11}b_{2q} & \cdots & \cdots & a_{1n}b_{21} & a_{1n}b_{22} & \cdots & a_{1n}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & a_{11}b_{p2} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{p1} & a_{1n}b_{p2} & \cdots & a_{1n}b_{pq} \\ \vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & \ddots & \vdots & \vdots & & \vdots \\ a_{m1}b_{11} & a_{m1}b_{12} & \cdots & a_{m1}b_{1q} & \cdots & \cdots & a_{mn}b_{11} & a_{mn}b_{12} & \cdots & a_{mn}b_{1q} \\ a_{m1}b_{21} & a_{m1}b_{22} & \cdots & a_{m1}b_{2q} & \cdots & \cdots & a_{mn}b_{21} & a_{mn}b_{22} & \cdots & a_{mn}b_{2q} \\ a_{m1}b_{p1} & a_{m1}b_{p2} & \cdots & a_{m1}b_{pq} & \cdots & \cdots & a_{mn}b_{p1} & a_{mn}b_{p2} & \cdots & a_{mn}b_{pq} \end{bmatrix} \quad (\mathbf{3.9})$$

$\otimes$ işlemi Kronecker çarpımını göstermek üzere Hadamard dönüşümü T_H, T_{Odd} ve T_{OddOdd} şu şekilde tanımlanır (**3.10**):

$$\begin{aligned} T_H &= H \otimes H \\ T_{Odd} &= T_H \otimes T_R \\ T_{OddOdd} &= T_R \otimes T_R \end{aligned} \quad (\mathbf{3.10})$$

FCT dönüşümün ilk basamağı Şekil 3.3'te de görüldüğü gibi 4 adet 2x2 Hadamard dönüşümünden (T2x2h) oluşmaktadır. İlk olarak 4x4 bloğun köşelerine (a, b ,c , d), ardından merkeze (e, f ,g , h), merkezden sonra alt ve üst sınırlara (i, l ,m , n) ve son olarak da sağ ve sol sınırlara (o, p, q , r) T2x2h uygulanır.

İkinci basamakta da sırasıyla ilk olarak sol üst kısma (A, B, C, D) T2x2h, ardından sağ üst kısma (E, F, G, H) TOdd, daha sonra sol alt kısma (I, L, M, N) TOdd ve son olarak sağ alt kısma (O, P, Q, R) TOddOdd uygulanır. Buradaki sıralamalar önemlidir. TOdd ve TOddOdd için kod taslakları sırasıyla Çizelge 3.2 ve Çizelge 3.3'te gösterilmiştir.

İkinci basamak da tamamlandıktan sonra katsayılar "İleri Permütasyon" fonksiyonu aracılığıyla Çizelge 3.4'te görüldüğü gibi tekrardan sıralanır.

3.2.4.2 OT (Overlap Transform)

İsteğe bağlı olarak FCT'den önce OT uygulanabilir. FCT sadece 4x4 piksellik blokların içinde uygulandığı ve komşuluklarıyla ilgilenmediği için bloklanma kaçınılmazdır. OT, bloklanma hatalarını kısıtlayabilmek amacıyla tasarlanmış bir ön filtredir. OT, sadece makro-bloklara değil, aynı zamanda komşu makro-blokların sınırlarına da uygulanır. Bu anlamda hem işlem karmaşıklığı hem de hafıza kullanımı daha fazladır. OT operatörünün işlevselleğini OVERLAP_MODE parametresi belirler.

Çizelge 3.2: TOdd operatörü [1].

TOdd kod taslağı
f = b - c
e = a + d
g = c + ((f + 1)/2)
h = (e + 1)/2 - d
T1 = f - ((3*e + 4)/8)
T2 = e + ((3*f + 4)/8)
T3 = h - ((3*g + 4)/8)
T4 = g + ((3*h + 4)/8)
D = T3 + (T1/2)
C = T4 - ((T2+1)/2)
B = T1 - T3
A = T2 + T4

Çizelge 3.3: TOddOdd operatörü [1].

TOddOdd kod taslağı
f = -b
g = -c
h = d + a
T1 = h/2
T2 = g/2
V1 = g - f
e = a - T1
V2 = f + T2
V3 = e + ((V2*3 + 4)/8)
V4 = V2 - ((V3*3 + 3)/4)
V5 = V3 + ((V4*3 + 3)/8)
D = V4 - T1
C = V5 + T2
B = V1 + V4
A = h - V5

Çizelge 3.4: İleri permütasyon.

i:	0	1	2	3	4	5	6	7
İleri Permütasyon[i]:	0	8	4	6	2	10	14	12
i:	8	9	10	11	12	13	14	15
İleri Permütasyon[i]:	1	11	15	13	9	3	7	5

Bu parametre 0, 1 ve 2 değerliklerini alabilir. Bu üç durum şu şekilde sıralanır:
- OVERLAP_MODE = 0: Çakışan ön-filtrelemenin yapılmadığı durumdur. Genellikle kayıpsız kodlama için en uygundur.
- OVERLAP_MODE = 1: 1 seviye çakışan ön-filtreleme olarak adlandırılır. Kodlanmış 4x4 piksellik blokların değerleri komşulukları göz önüne alınarak değiştirilir. Değişen görüntüler ve kuantalama seviyeleri için genelde en kısa bit dizisini üretir.
- OVERLAP_MODE = 2: 2 seviye çakışan ön-filtreleme olarak adlandırılır. Önce 1 seviye çakışan ön-filtreleme uygulanır. Ek olarak, kodlanmış 16x16 piksellik makro-blokların değerleri komşulukları göz önüne alınarak değiştirilir. Yüksek kuantalama seviyeleri için önerilir fakat işlem karmaşıklığı fazladır.

OT'yi belirleyen 4 filtre bulunmaktadır: 4x4 ön-filtre, 4-nokta ön-filtre, 2x2 ön-filtre ve 2-nokta ön-filtre. Bu filtrelerin kullandığı operatörler olan 2x2T_h_Enc(), fwd_Rotate(), fwd_Scale() ve fwdT_Odd_Odd() operatörleri ve kod taslakları sırasıyla Çizelge 3.5, Çizelge 3.6, Çizelge 3.7 ve Çizelge 3.8'de verilmiştir.

Çizelge 3.5: 2x2T_h_Enc() kod taslağı [2].

2x2T_h_Enc(a,b,c,d)
a += d
b -= c
t1 = d
t2 = c
c = ((a – b) » 1) – t1
d = t2 + (b » 1)
b += c
a -= (d * 3 + 4) » 3

Çizelge 3.6: fwd_Rotate() kod taslağı [2].

fwd_Rotate(a,b)
b -= (a + 1) » 1
a += (b + 1) » 1

4x4 ön-filtreleme öncelikle OVERLAP_MODE 1 veya 2 olan durumlarda tüm blok kesişimlerinde uygulanır. Bunun yanında OVERLAP_MODE'nin 2 olduğu durumlarda DC-LP dizisinin tüm blok kesişimlerinde de uygulanır. OVERLAP_MODE'nin 2 olduğu ve YUV 4:2:0 ve 4:2:2 seçili durumlarda ise parlaklık (luma) düzlemine de

Çizelge 3.7: fwd_Scale() kod taslağı [2].

fwd_Scale(a,b)
b -= (a * 3 + 0) » 4
a -= (b * 3 + 0) » 3
b = (a » 1) - b
a -= b

Çizelge 3.8: fwdT_Odd_Odd() kod taslağı [2].

fwdT_Odd_Odd(a,b,c,d)
d += a
c -= b
t1 = d » 1
t2 = c » 1
a -= t1
b += t2
a += (b * 3 + 4) »3
b -= (a * 3 + 2) » 2
a += (b * 3 + 6) » 3
b -= t2
a += t1
c +=b
d -= a

uygulanan bir filtredir. 4x4 ön-filtreleme, 4x4PreFilter (a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p) olarak Çizelge 3.9'da verilmiştir.

Doğrusal 4-nokta ön-filtre, görüntünün sınırlarındaki kenar komşulukları olan 2x4 ve 4x2'lik alanlara uygulanır. 4-nokta ön-filtre, 4PreFilter(a, b, c, d) olarak Çizelge 3.10'da verilmiştir.

2x2 ön-filtre, YUV 4:2:0 ve YUV 4:2:2 renk bileşenlerinden oluşan DC-LP dizisinin blok komşuluklarında uygulanır. 2x2 ön-filtre, 2x2PreFilter(a, b, c, d) olarak Çizelge 3.11'de verilmiştir.

2-nokta ön-filtre blok komşuluğuna sahip 2x1 ve 1x2'lik sınırlarda uygulanır. 2-nokta ön-filtre, 2PreFilter(a, b) olarak Çizelge 3.12'de verilmiştir.

3.2.5 Kuantalama

FCT dönüşüm işleminin ardından dönüşüm katsayılarının kuantalama değeri (QP) adı verilen bir sayıya bölündüğü basamaktır. Kayıpsız kodlamada QP 1'e eşitken,

Çizelge 3.9: 4x4 ön-filtre kod taslağı [2].

4x4PreFilter (a, b,..., p)
2x2T_h_Enc(a, d, m, p);
2x2T_h_Enc (b, c, n, o);
2x2T_h_Enc (e, h, i, l);
2x2T_h_Enc (f, g, j, k);
fwd_Scale (a, p);
fwd_Scale (b, o);
fwd_Scale (e, l);
fwd_Scale (f, k);
fwd_Rotate (n, m);
fwd_Rotate (j, i);
fwd_Rotate (h, d);
fwd_Rotate (g, c);
fwdT_Odd_Odd(k, l, o, p);
2x2T_h (a, m, d, p, 0);
2x2T_h (b, c, n, o, 0);
2x2T_h (e, h, i, l, 0);
2x2T_h (f, g, j, k, 0);

Çizelge 3.10: 4-nokta ön-filtre kod taslağı [2].

4PreFilter(a, b, c, d)
a –= ((d * 3 + 16) » 5);
b –= ((c * 3 + 16) » 5);
d –= ((a * 3 + 8) » 4);
c –= ((b * 3 + 8) » 4);
a += d – ((d * 3 + 16) » 5);
b += c – ((c * 3 + 16) » 5);
d –= ((a + 1) » 1);
c –= ((b + 1) » 1);
fwd_Rotate(c, d);
d += ((a + 1) » 1);
c += ((b + 1) » 1);
a –= d;
b –= c;

Çizelge 3.11: 2x2 ön-filtre kod taslağı [2].

2x2PreFilter(a, b, c, d)
a += d;
b += c;
d –= ((a + 1) » 1);
c –= ((b + 1) » 1);
b –= ((a + 2) » 2);
a –= ((b + 1) » 1);
b –= ((a + 2) » 2);
d += ((a + 1) » 1);
c += ((b + 1) » 1);
a –= d;
b –= c;

Çizelge 3.12: 2-nokta ön-filtre kod taslağı [2].

2PreFilter(a, b)
b –= ((a + 4) » 3);
a –= ((b + 2) » 2);
b –= ((a + 4) » 3);

kayıplı kodlamada 1'den büyük bir değerlik alır. JPEG XR standardı 3 farklı QP tipine sahiptir: renk düzleminde, frekans bantlarında ve uzamsal boyutta.

3.2.5.1 Renk düzleminde kuantalama

JPEG XR standardı renk düzleminde kuantalama sırasında bazı esnekliklere sahiptir:

1. Tüm renk düzlemleri için aynı QP kullanılabilir.

2. Parlaklık (Luma) renk düzlemi için farklı bir QP, diğer tüm renk düzlemleri için aynı QP kullanılabilir.

3. Her renk düzlemi için farklı QP kullanabilir.

3.2.5.2 Frekans bantlarında kuantalama

JPEG XR standardı frekans bantlarında kuantalama sırasında bazı esnekliklere sahiptir:

1. Tüm frekans bantlarında aynı QP kullanılabilir.

2. DC ve HP bantlarda aynı, LP bandında farklı QP kullanılabilir.

3. LP ve HP bantlarda aynı, DC bandında farklı QP kullanılabilir.
4. Tüm frekans bantlarında farklı QP kullanılabilir.

3.2.5.3 Uzamsal boyutta kuantalama

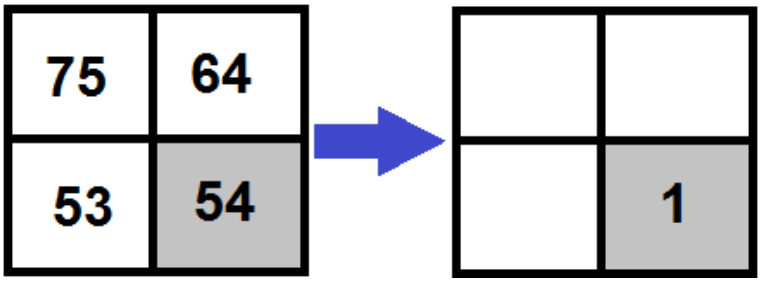
JPEG XR standardı uzamsal boyutta kuantalama sırasında bazı esnekliklere sahiptir:
1. Tüm görüntü için aynı QP kullanılabilir.
2. Her karo (tile) için farklı QP kullanılabilir.
3. Aynı karo (tile) içerisindeki farklı makro-bloklar için farklı QP kullanılabilir.

3.2.6 Adaptif Tahmin

Çoğu zaman bir görüntüde küçük bir alandaki piksel veya bloklar benzerlik gösterir. Yani görüntünün görece küçük alanlarında büyük bir tekrar söz konusudur. Bu yüzden aynı değerleri tekrar tekrar kaydetmek yerine sadece değişen değerler için farkları tutmak daha verimlidir. Böylece entropi kodlama için daha fazla 0 elde edilmiş olur. Tahminde asıl amaç asıl katsayı ile komşuluğu arasındaki farkı alıp bu farkı kaydetmektir. JPEG XR standardında adaptif tahmin kullanılır. Bu tahmin sadece iki komşu makro-bloğun büyük bir benzerliğe sahip olduğu durumlarda işler. Adaptif olmasının nedeni ise tahmin yönünün komşu makro-bloklardaki benzerlik miktarlarına göre değişmesinden kaynaklanmaktadır. Tahmin yönü soldan, üstten veya sol-üstten olabilir. Aynı zamanda tahmin tarzı farklı frekans bantlarında farklılaşır.

3.2.6.1 DC Tahmin

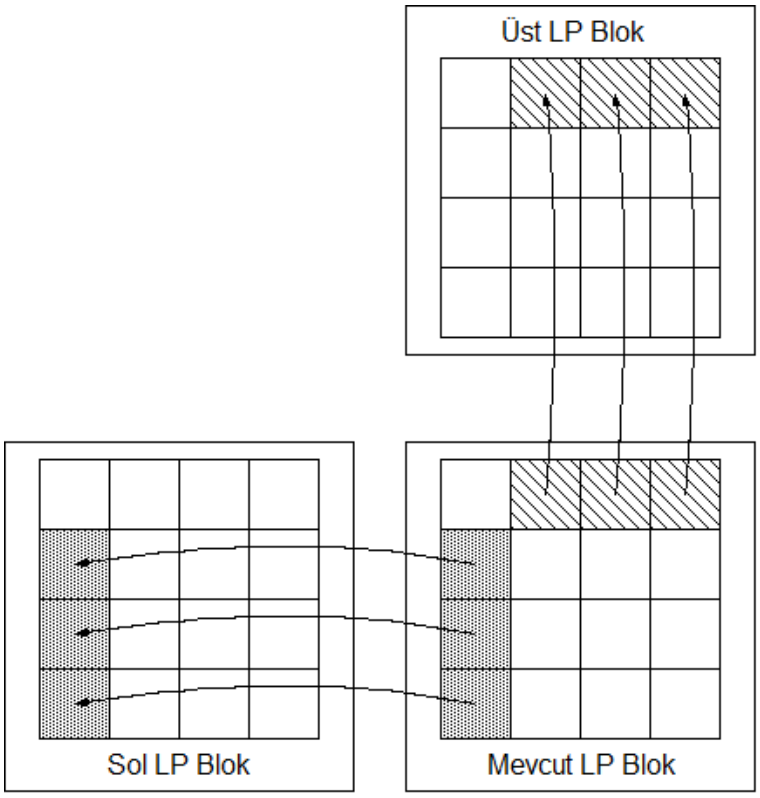
DC tahmin makro-bloklar arasında gerçekleşir. Soldan, üstten veya sol-üstten olabilir. En fazla benzerlik gösteren makro-bloğun DC seviyesi seçilir ve mevcut makro-bloktakiyle farkı alınır. Şekil 3.4'te DC tahminin makro-bloklar arasında nasıl gerçekleştiği bir örnekle gösterilmiştir. Her makro-blokta bir adet DC katsayı olduğu için, dört DC katsayisi dört makro-bloğa karşılık gelmektedir. Örnekte ilgili makro-bloğun DC katsayısı 54'tür. Komşuluklarına bakıldığında ise en büyük benzerliğe mevcut makro-bloğun solundaki makro-blok sahiptir. Mevcut makro-blok ile benzerliği en yüksek olan makro-bloğun farkı alınır ve sonuç 1 olarak bulunur. Bu yeni değer bir sonraki basamağa (adaptif tarama) geçecek olan değerdir.



Şekil 3.4: DC tahmin örneği [3].

3.2.6.2 LP Tahmin

DC tahminde olduğu gibi LP tahmin de makro-bloklar arasında gerçekleşmektedir. Geçerli iki tahmin yönü vardır: sol ve üst. LP tahmin yönü DC tahmin modu tarafından bahsi geçen iki makro-bloğun quantalama parametreleri aynı ise belirlenir, aksi takdirde atlanır. DC tahmin yönü sol-üst ise LP tahmin yapılmaz, atlanır. Şekil 3.5'te LP tahminin nasıl yapıldığı gösterilmiştir.



Şekil 3.5: LP tahmin [3].

3.2.6.3 HP Tahmin

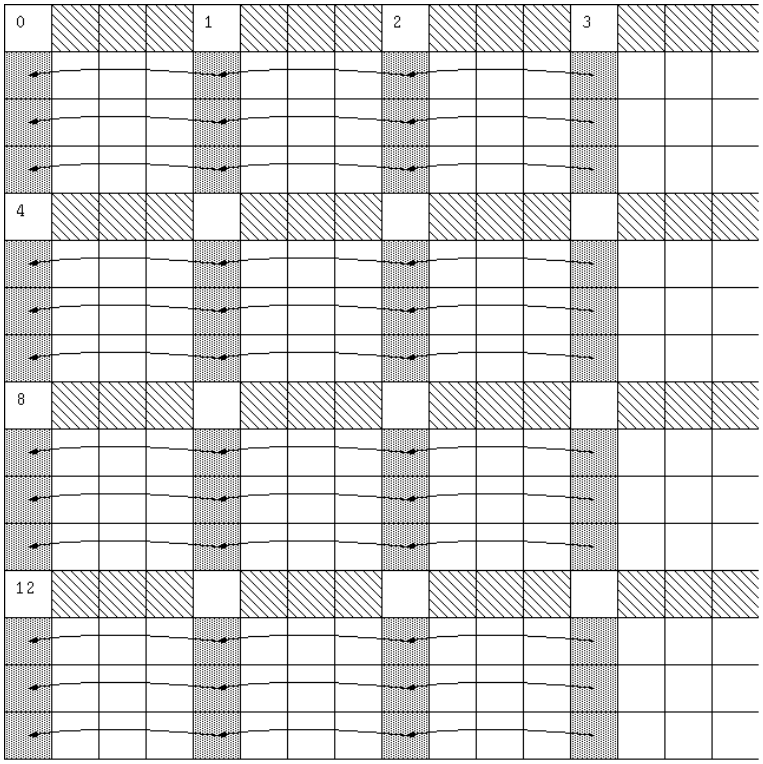
DC ve LP tahminden farklı olarak HP tahmin her makro-bloğun kendi içerisinde gerçekleşir. Amaç, makro-bloğu oluşturan bloklar içerisindeki farklılıkları ortaya çıkarmak ve kaydetmektir. Tahmin yönleri sol ve üsttür. Bu yönler mevcut makro-bloktaki LP katsayıları tarafından belirlenir. Şekil 3.6'da HP tahminin nasıl yapıldığı gösterilmiştir.

3.2.7 Entropi Kodlama

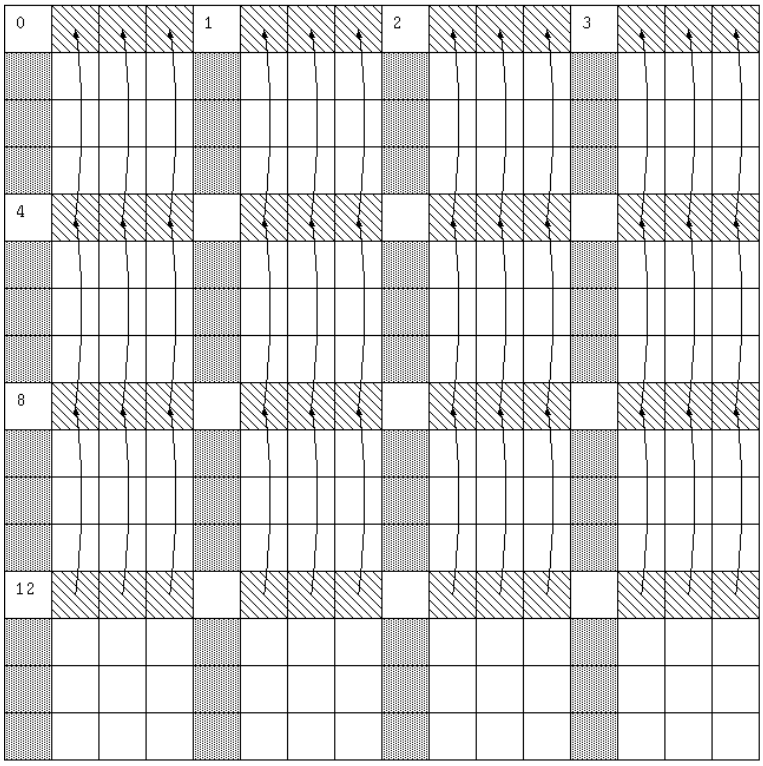
Kuantalama basamağından sonra daha verimli sıkıştırmayı sağlabilmek için kuantalanan diziler entropi kodlayıcı basamağına iletilirler. Entropi kodlama basamağı matematiksel algoritmalara dayanmaktadır ve birbirine benzer alanlardaki benzerliği kullanarak daha fazla sıkıştırma sağlamak üzerine temellendirilmiştir. En bilinen örneği ise ardışık benzer sembollerin oluşturduğa gruba bir kod ya da sembol atayarak gerçeklenen RLE (Run-Level Encoding) algoritmasıdır.

JPEG XR kodlama algoritmasında ise farklı bir katsayı kodlama kullanılır. Önce elde edilen bit dizileri normalize edilir ve normalizasyon sonrası bu bitler sabit uzunluklu FLEXBITS olarak kodlanır. Birleşik sembollerden ilki sıfır olmayan katsayıları ve diğeri ise ondan sonra gelen sıfır dizileri şeklindedir. JPEG XR kodlama algoritmasında VLC (Variable-Length Coding) tabloları önceden kodlanmış sembollerin istatistikleri kullanılarak adapte edilmiştir ve sıklıkla kullanılan sembollere daha kısa kod sözcükleri atanmıştır. VLC tablolarının adaptasyonu iki şekilde olabilirdi:
İleri adaptasyon: VLC tabloları önceden tanımlanır ve her defasında bitdizisiyle birlikte gönderilir. Çok fazla yük getirdiği için nadiren kullanılabilir.
Geri adaptasyon: VLC tabloları önceden kodlanmış sembollerin istatistikleri kullanılarak ayarlanır, yük getirmez fakat çok fazla işlem karmaşıklığına yol açar.

JPEG XR kodlama algoritması bu iki metodun dezavantajlarından kaçınmak yeni bir metod önerir. Bu metodda geniş kapsamlı istatistikler kullanılarak oluşturulmuş az sayıda VLC tablosu bulunur. Entropi kodlama sırasında henüz kodlanan sembollere göre en uygun kod tablosu seçilir. Entropi kodlama sırasında T_i kod tablosu



a) Soldan HP tahmin



b) Üstten HP tahmin

Şekil 3.6: HP tahmin [3].

kullanılıyor ise geçiş yapılabilecek diğer kod tablosu seçenekleri T_{i-1} ve T_{i+1} olarak belirlenmiştir. İlk değerleri 0 olan, adlarına diskriminant denilen $D1$ ve $D2$ değişkenleri hangi kod tablosunun seçilmesinin avatanjlı olduğunu ölçmede kullanılır. Herhangi bir sembolün kodlanmasında T_{i+1} kod tablosu T_i kod tablosuna göre daha avantajlı ise $D1$ artırılır, aksi durumda da azaltılır.

Sonuç olarak JPEG XR kodlama algoritmasının kullanmış olduğu uyarlamalı VLC tablosu algortiması işlem karmaşıklığı açısından bilinen VLC tablosu algoritmasından daha az karmaşıktır.

3.3 JPEG XR'nin JPEG ile Yapısal Karşılaştırılması

JPEG XR'nin JPEG ile karşılaştırıldığında üstün gelen noktalarını şu şekilde sıralayabiliriz [15]:

1. Daha iyi sıkıştırma: JPEG XR, JPEG ile karşılaştırıldığında benzer kalite seviyesine sahip imajlarda daha yüksek sıkıştırma oranları sağlamaktadır.
2. Kayıpsız kodlama: JPEG XR'nin kayıpsız kodlama desteği bulunmaktadır.
3. Karolara ayırma desteği: JPEG XR kodlanmış bir görüntü farklı karolara ayrılabilir ve her bölgenin kodu ayrı ayrı çözülebilir. Bu sayede bütün bir görüntünün kodunu çözmeden ilgilenilen bölgenin kodu çözülebilir.
4. Daha fazla renk hassasiyeti: JPEG piksel başına gri ölçekte 8-bit tam sayı, renkli görüntülerde ise 24-bit (gerçek renk) tam sayıya kadar renk derinliği sunarken; JPEG XR piksel başına 64-bit tam sayı ve 128-bit kayan sayı renk derinliği sunabilmektedir.
5. Alfa kanal desteği: JPEG XR'nin şeffaflığı ifade etmede kullanılan alfa kanal desteği bulunmaktadır.
6. Üstveri (Metadata) desteği: Bir JPEG XR dosyası opsiyonel olarak, farklı cihazlarda renklerin tutarlı bir şekilde temsil edilmesi için ICC renk profili barındırabilir. JPEG XR ayrıca Exif ve XMP üstveri formatlarını da desteklemektedir.
7. JPEG/JFIF'nin RGB'den YCbCr renk uzayına dönüşümü sırasında doğrusal dönüşüm kullanıldığı için yuvarlama hatalarına bağlı kayıplar oluşmaktadır. JPEG XR'de ise RGB'den YUV renk uzayına dönüşümler kayıpsızdır.
8. JPEG kodlama esnasında 8x8'lik blok boyutu kullanırken, JPEG XR'de 4x4'lük blok boyutu kullanılmaktadır. Bu durum JPEG XR kodlamada frekans dönüşümlerinin

daha incelikli olmasını sağlar.

9. JPEG'in kullandığı frekans dönüşümü olan DCT, yuvarlama hatalarından dolayı az da olsa kayıplı bir dönüşümdür. JPEG XR'nin kendi frekans dönüşümü ise tam sayı tabanlıdır ve tam olarak terslenebilir olduğundan kayıpsızdır.

10. Entropi kodlama basamakları karşılaştırıldığında JPEG XR'nin daha uyarlamalı ve bunun sonucunda daha karmaşık olduğu görülmektedir. Örneğin JPEG sabit zig-zag sıralama kullanırken, JPEG XR uyarlamalı katsayı yeninden sıralama tekniğini kullanmaktadır (İleri permütasyon fonksiyonu).

4. JPEG XR - JPEG - JPEG2000 KODLAMALARININ PERFORMANS KARŞILAŞTIRMASI

JPEG XR ve JPEG kodlama algoritmalarının performanslarının karşılaştırılabilmesi için sırasıyla JPEG XR Referans Yazılım [14] ve Independent JPEG Group tarafından sunulan jpeg-9 [16] versiyon yazılımlar kullanılmıştır. Performans karşılaştırmalarında bir ölçüt olması açısından JPEG-2000'in kodeklerinden biri olan JPEG-2000 Bölüm 1 [6] standardının bir referans uygulaması olan JASPER [17] kullanılmıştır. Kullanılan kodekler ve sürümleri Çizelge 4.1'de, kodek ayarları Çizelge 4.2'de ve kullanılan sıkıştırılmamış test görüntüleri Çizelge 4.3'te verilmiştir.

Çizelge 4.1: Kodekler

Kodek	Sürüm
JPEG	Jpeg-9
JPEG XR	ITU-T Rec. T.835 Referans yazılım
JPEG-2000	JASPER-1.900.1

Çizelge 4.2: Kodek ayarları

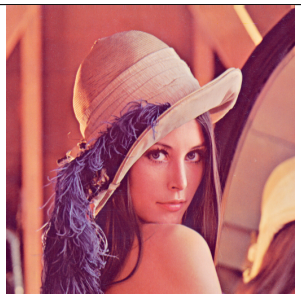
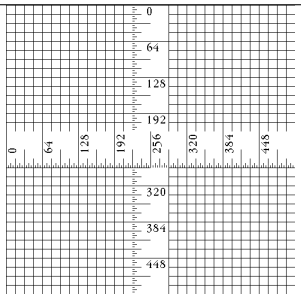
Kodek	Parametre	Değerler
Jpeg-9	quality	0-100 (5-95 önerilen)
JPEG XR Referans Yazılım	q	0-255
	l	0, 1, 2
	f	YUV444, YUV422, YUV420
JASPER	rate	0-1

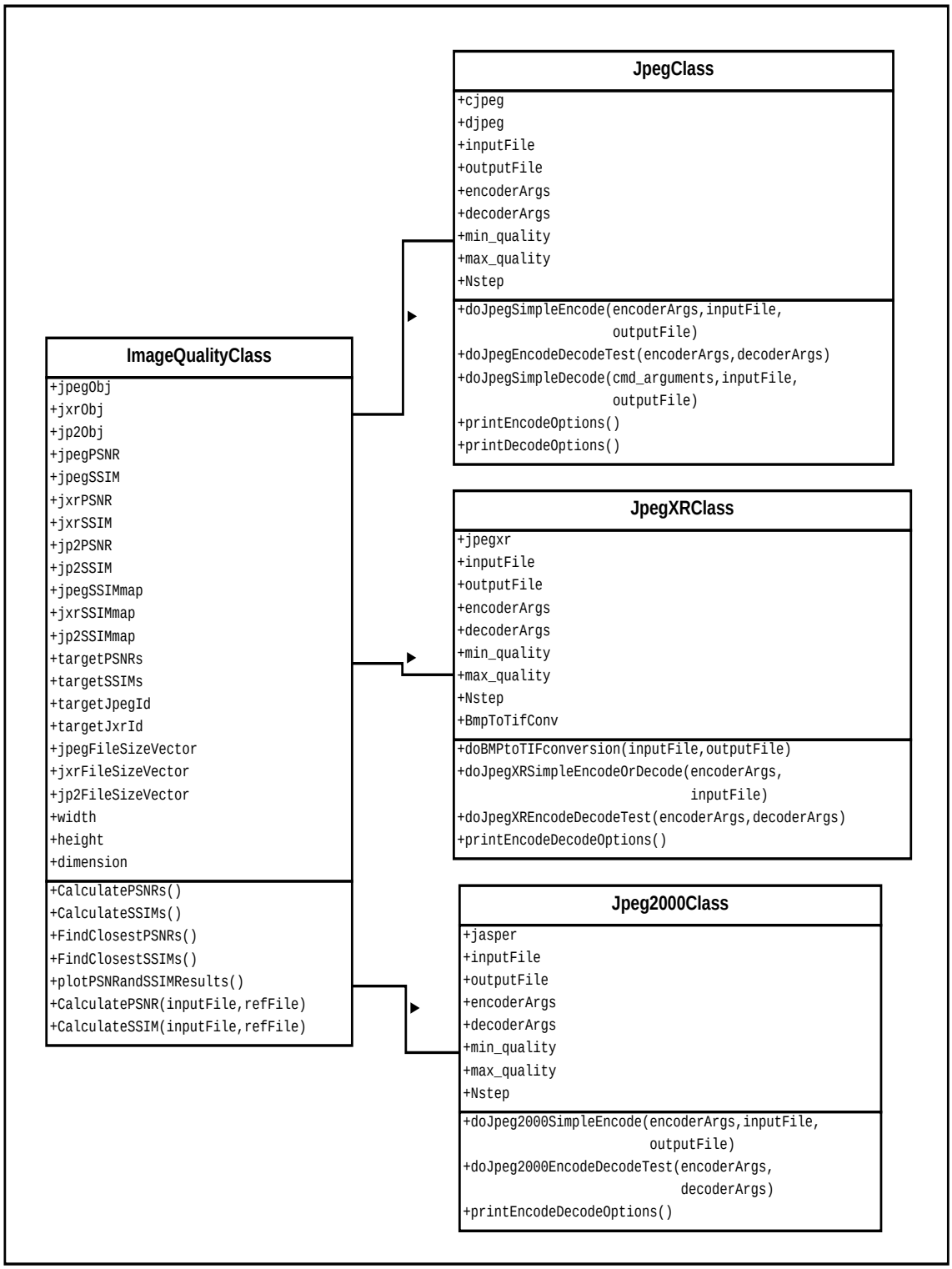
Elde edilen kodlama ve kod çözme programları MATLAB R2011b versiyon programı için yazılmış sınıflar aracılığıyla sarmalanmış ve nesne yönelimli bir tasarım yapılarak istenilen tüm sonuçların elde edilmesi sağlanmıştır. Bu çalışmada yazılan sınıflar Ekler bölümünde verilmiştir. Sınıf diagramları Şekil 4.1'de verilmiştir.

4.1 Nesnel Karşılaştırma ve Sonuçları

Nesnel karşılaştırma, farklı görsel ve yapısal özelliklere sahip test görüntülerinin JPEG XR, JPEG ve JPEG-2000 kodekleri ile farklı kalite seviyelerinde kodlanmasını,

Çizelge 4.3: Test görüntüleri.

Görüntü	Boyut (byte)	Çözünürlük	Bit derinliği
 lenna.bmp	786,486	512x512	24-bit
 barbara.bmp	983,094	640x512	24-bit
 baboon.bmp	786,486	512x512	24-bit
 lighthouse.bmp	983,094	512x640	24-bit
 ruler.bmp	786,486	512x512	24-bit

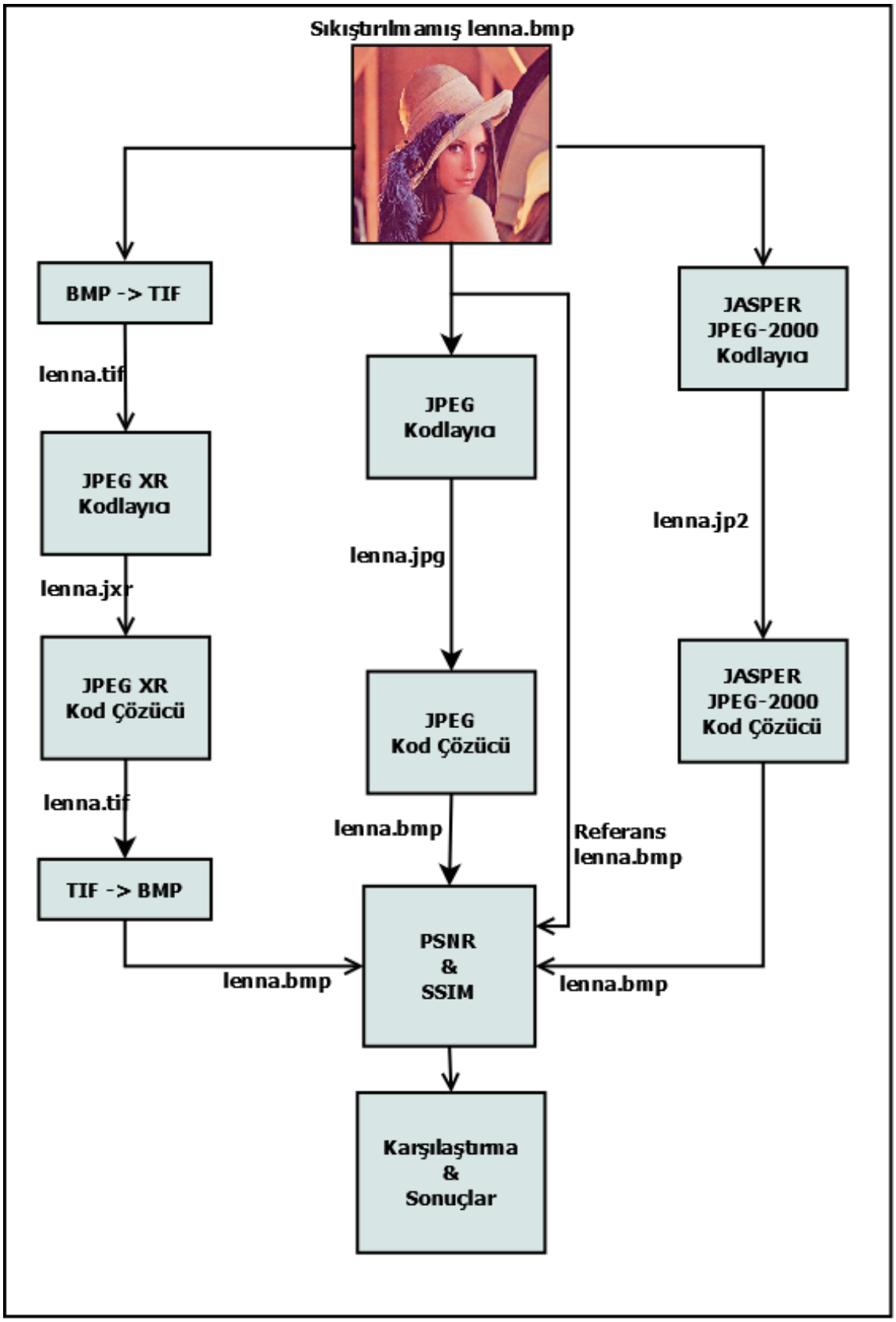


Şekil 4.1: JPEG, JPEG XR, JPEG-2000 ve ImageQuality sınıf diagramları.

kodlanan görüntülerin PSNR (Tepe işaret gürültü oranı) ve SSIM (Yapısal benzerlik) gibi performans metrikleri ile ölçülmesini ve elde edilen sonuçların karşılaştırılmasını ifade etmektedir.

Öncelikle test görüntü kümesinden seçilen bir görüntü için JPEG kodlamanın 0-98 arası kalite seviyelerinde birer birim artış ile (quality:1, 2, 3,..., 98) farklı kalite seviyelerinde kodlama yapılacak ve JPEG kodlanmış görüntüler kaydedilecektir. JPEG kodlanmış görüntüler JPEG kod çözücü ile nesnel karşılaştırmada kullanılmak üzere tekrardan bmp formatına dönüştürülecektir. JPEG XR kodlama için aynı test görüntüsü önce IrfanView 4.36 [18] versiyon yazılım ile bmp formatından tif formatına dönüştürülecek ve JPEG XR kodlayıcı ile 0-100 arası kalite seviyelerinde birer birim artışlar ile (q:1, 2, 3.., 100) kodlanacaktır. JPEG XR'nin kalite parametresi JPEG'in kalite parametresine göre ters çalışmaktadır. Kalite parametresinin değerliği arttıkça sıkıştırma oranı artmakta ve kalite düşmektedir. Yani JPEG XR kodlamada 0'dan uzaklaştıkça kodlanmış görüntü kalitesi düşmektedir. JPEG XR kodlama için kalite skalası 0-255 arasındadır, bu çalışmada 0-100 arası kalite skalası karşılaştırma için yeterlidir. Fakat bazı görüntüler için diğer kodeklerle karşılaştırıabilir PSNR ve SSIM değerliği verdiği görülürse 0-100 skalası 0-110 gibi revize edilebilir. JPEG XR kodlanmış görüntüler de nesnel karşılaştırmada kullanılmak üzere önce JPEG XR kod çözücü ile tif formatına, ardından da yine IrfanView 4.36 [18] versiyon yazılım ile bmp formatına dönüştürülecektir. JPEG-2000 kodlama için ise test görüntüsü 0-0.35 arası sıkıştırma oranları arasında yaklaşık 0,003 birimlik artışlar ile JASPER kodlanacak ve ardından nesnel karşılaştırma için tekrardan JASPER kod çözücü kullanılarak bmp formatına dönüştürülecektir. Elde edilen JPEG XR, JPEG ve JPEG-2000 kodlanmış görüntülerin, referans bmp formatlı görüntü baz alınarak PSNR ve SSIM değerlikleri hesaplanacak ve oluşan sıkıştırılmış dosyaların bit oranları da elde edilerek kodekler arası karşılaştırma yapılacaktır. Önerilen metot Şekil 4.2'de verilmiştir.

JPEG XR kodlamada görüntü kalitesine etki eden iki temel parametre bulunmaktadır: -q (kalite) ve -l (çakışan blok filtrelemesi). "-l" parametresi JPEG XR dönüşümü sırasında FCT'den önce uygulanabilen opsiyonel bir filtredir. Kodlama esnasında "-l" parametresinin 0 olduğu durumda bu filtre uygulanmaz, işlem karmaşıklığı azalır fakat kodlanmış görüntülerde bloklanma etkisi görülür. Bu parametrenin 2 olduğu



Şekil 4.2: JPEG XR, JPEG ve JPEG-2000 performans karşılaştırması için önerilen metot.

durumda ise kodlanmış görüntüde çok fazla bulanıklaşma gözlemlenmektedir. "-l" parametresinin görüntü kalitesine etkisi Çizelge 4.4'te verilmiştir. Test görüntüsü üç farklı "-l" parametresi (0, 1, 2) ile aynı kalite seviyesinde (q = 72) JPEG XR sıkıştırma yapılmış ve elde edilen görüntülerin Y-PSNR ve SSIM değerlikleri MSU Quality Measurement Tool programının ticari olmayan 3.0 versiyonu [19] ile hesaplanmıştır. "-l" parametresi 0 olduğu zaman aşırı miktarda bloklanma olduğu, 2 olduğunda çok fazla bulanıklaşma gözlemlendiği için ve en iyi SSIM değeri "-l" parametresi 1 olduğunda elde edildiğinden bu çalışmada "-l" parametresi daima 1 olarak kullanılmıştır.

Çizelge 4.4: JPEG XR kodlama -l (çakışan blok filtreleme) parametresi.

l = 0 q = 72	l = 1 q = 72	l = 2 q = 72
		
Y-PSNR = 32.20 SSIM = 0.8727 bpp = 0.356	Y-PSNR = 31.87 SSIM = 0.8786 bpp = 0.296	Y-PSNR = 31.39 SSIM = 0.8782 bpp = 0.291

PSNR ve SSIM gibi metrikler kullanılırken aynı formata sahip görüntülerin karşılaştırılması, karşılaştırmanın objektifliği açısından önemlidir. Bu yüzden PSNR ve SSIM ölçümleri yapılırken referans test görüntüsü ve karşılaştırılan görüntülerin aynı formatta (bmp) olmasına dikkat edilmiştir. JPEG XR kodlarken ve kod çözerken sıkıştırılmamış bmp görüntülerden tif formatına geçişlerde bir kayıp söz konusu değildir. MATLAB'da, sıkıştırılmamış bmp görüntüsü ile bu görüntüden dönüştürülmüş tif görüntüleri MATLAB komutları ile dizilere aktarılmış ve bu diziler

karşılaştırıldığında birebir aynı oldukları gözlemlenmiştir. PSNR ve SSIM hesapları bu dizileri kullandığı için objektiflik sağlanmış olmaktadır.

4.1.1 Tepe işaret gürültü oranı (PSNR)

PSNR bir işaretin olası maksimum gücünün işaret üzerindeki gürültünün gücüne oranını gösteren bir mühendislik terimidir. İşaretlerin çok çeşitlilik gösterebilmesinden ötürü birimi logaritmik ölçekte decibel (dB) ile ifade edilir.

PSNR genellikle kayıplı sıkıştırma yapılmış görüntülerin kalitesi hakkında bir ölçüm yapılmasına olanak veren bir metriktir. İşaret orijinal veriyi temsil ederken gürültü ise sıkıştırma kaynaklı hatayı temsil etmektedir. Sıkıştırma kodlamalarını karşılaştırırken PSNR aslında insanın kalite algısına bir yaklaşım olarak düşünülebilir. Genelde yüksek PSNR değeri sıkıştırma sonrası oluşan görüntünün daha yüksek kaliteye sahip olduğunu söylese de bazı durumlarda böyle olmayabilir.

PSNR çoğunlukla MSE (Ortalama karesel hata) vasıtasıyla ifade edilir. $m \times n$ bir tek-renkli görüntü I ve gürültülü hali K için MSE hesabı aşağıdaki şekilde verilmektedir (**4.1**).

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{4.1}$$

PSNR ise şu şekilde hesaplanmaktadır (**4.2**).

$$\begin{aligned} PSNR &= 10 \log_{10} \left(\frac{{MAX_I}^2}{MSE} \right) \\ &= 20 \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \\ &= 20 \log_{10} (MAX_I) - 10 \log_{10} (MSE) \end{aligned} \tag{4.2}$$

Burada, MAX_I görüntüde bir pikselin alabileceği maksimum değerliği belirtmektedir. Pikseller 8bit ile gösterildiğinde bu değer 255; B bit ile ifade edildiğinde ise $2^B - 1$ olarak ifade edilir. Renkli görüntülerde PSNR hesabı aynı şekilde hesaplanır, fakat MSE hesaplanırken tüm renk değerleri (R, G , B) hesaba katılır ve ortalama alırken görüntü boyutu ile birlikte 3'e de bölünür. Her kanal için ayrı ayrı PSNR değeri hesaplamak da alternatif olarak düşünülebilir.

4.1.2 Yapısal benzerlik (SSIM)

Yapısal benzerlik (SSIM) indeksi [20] görüntülerin birbirine benzerliğini ölçmek için kullanılan bir metottur. Bir görüntünün diğerine benzerliğinin ölçülmesi için referans olarak sıkıştırılmamış ya da gürültüsüz görüntü kullanılmaktadır. Daha önce anlatılmış olan PSNR ve MSE metotlarının insan göz algısına uygun olmadığı ispatlandığı için SSIM bu geleneksel metotları geliştirmek için tasarlanmıştır.

PSNR ve MSE algılanabilir hatayı tahmin eden yaklaşımlardır. Diğer taraftan SSIM görüntüdeki bozulmayı yapısal bilgideki algılanan değişiklik olarak ele alır. Yapısal bilgi ise uzamsal olarak yakın piksellerin bağımlılığının daha güçlü olduğunu söyler. Bu bağımlılıklar ise görüntüdeki nesnelerin yapıları hakkında önemli bilgi taşımaktadır.

Aynı $N \times N$ boyutlu görüntüler x ve y ile ifade edildiğinde bu görüntüler için SSIM indeksi aşağıdaki şekilde hesaplanmaktadır (**4.3**).

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (\mathbf{4.3})$$

μ_x: x'in ortalaması,
μ_y: y'nin ortalaması,
σ_x^2: x'in varyansı,
σ_y^2: y'nin varyansı,
σ_{xy}: x ve y'nin kovaryansı,
L, piksellerin alabileceği maksimum değerlik,
$k_1 = 0.01$ ve $k_2 = 0.03$ varsayılan değerlere sahiptirler.
$c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ parametreleri paydaların 0'a yaklaşıp formülün kararsızlaşmasını önlemektedir.

Görüntü kalitesinin ölçülebilmesi için bu formül sadece parlaklık (luma) üzerine uygulanır. SSIM indeksi -1 ve 1 arasında değerlik alır. İki görüntü aynı ise 1 değerliğini alır. Bu çalışmada bu algoritmanın Matlab ile yazılmış kodu kullanılmıştır [21].

4.1.3 PSNR/Bit derinliğ ve SSIM/Bit derinliği grafikleri

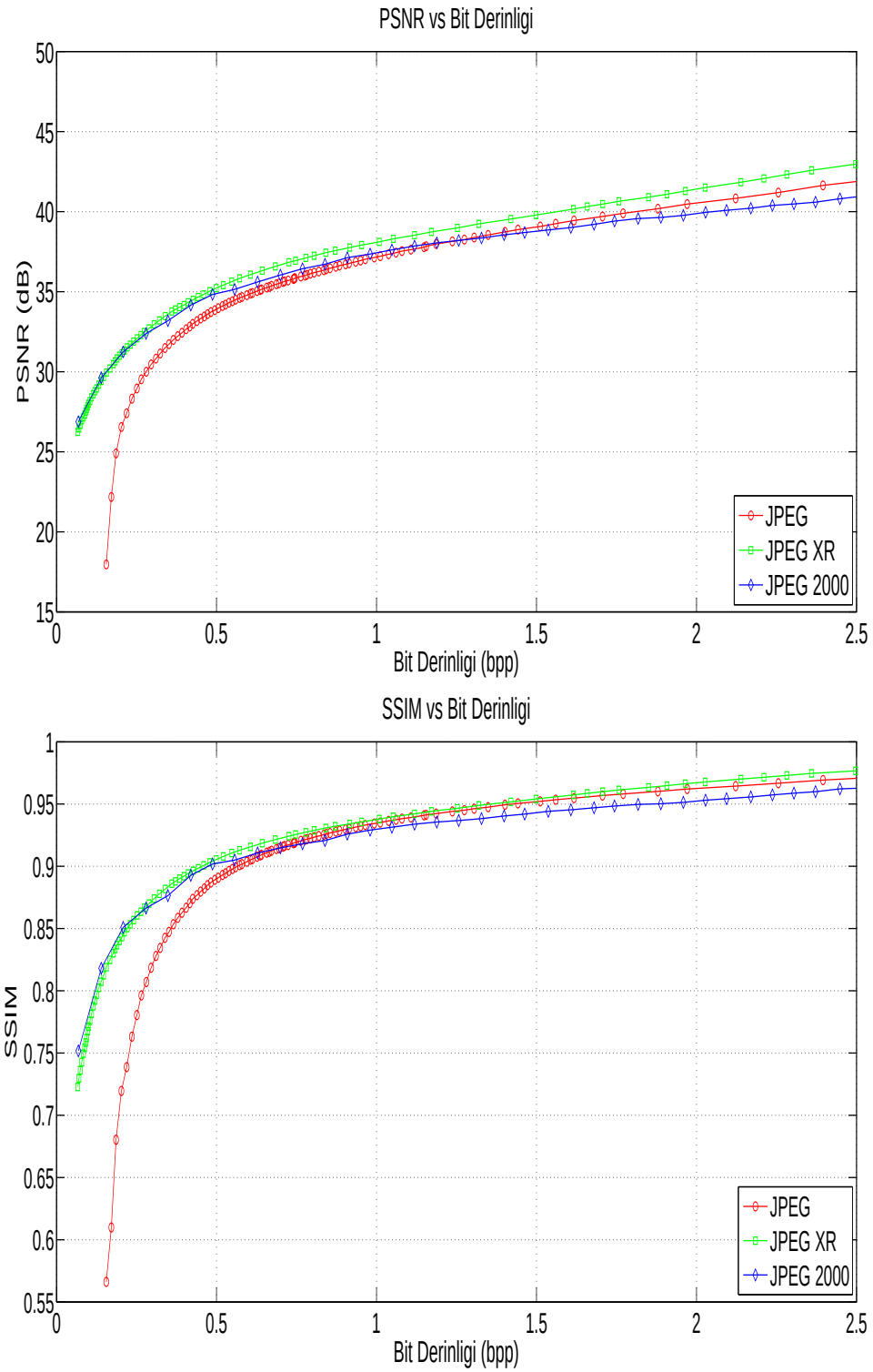
Bu bölümdeki grafikler sıkıştırma oranı ile sıkıştırma kalitesinin birbiriyle olan ilişkisini ortaya koyacaktır. Sıkıştırma kalitesinin ölçülmesinde PSNR ve SSIM metrikleri kullanılırken sıkıştırma oranı için sıkıştırılmış görüntülerin bit derinlikleri hesaplanmıştır. Burada bit derinliği kavramı, sıkıştırma sonrası piksel başına düşen bit sayısını karşılamaktadır ve bpp (piksel başına bit) ile ifade edilir. Herhangi bir I görüntüsü için S görüntünün bayt olarak dosya boyutuna, w görüntünün enine ve h görüntünün boyuna karşılık geldiğinde bpp değeri aşağıdaki şekilde hesaplanmaktadır (**4.4**). Bu değerin düşük olması sıkıştırma oranının yüksek olduğuna işaret eder. Örneğin 24-bit renk derinliğine sahip bir görüntünün sıkıştırılmış halinin 0.24 bpp değerine sahip olması, 1/100 oranında sıkıştırıldığının bir göstergesidir. Grafiklere 10 kat ve üzeri sıkıştırılmış görüntülerin kalite metrikleri yansıtılacaktır, bu da 2.4 bpp ve daha düşük bpp oranlarına karşılık gelmektedir.

$$BPP(I) = \frac{S \times 8}{w \times h} \tag{4.4}$$

4.1.3.1 Lenna

Şekil 4.3'te Lenna görüntüsü için elde edilen PSNR ve SSIM sonuçları verilmiştir. PSNR/Bit derinliği grafiği incelendiğinde yüksek sıkıştırma oranlarında JPEG'in JPEG XR ve JPEG-2000'in gerisinde kaldığı görülmektedir. JPEG XR ile JPEG 2000 karşılaştırıldığında ise yüksek sıkıştırma oranlarında birbirine çok yakın performans gösterirken, görece daha düşük sıkıştırma oranlarında JPEG-2000'in JPEG XR'nin gerisinde kaldığı görülmektedir.

SSIM/Bit derinliği grafiği incelendiğinde yine yüksek sıkıştırma oranlarında JPEG XR ve JPEG-2000'in JPEG'e göre çok daha iyi performans gösterdiği görülmektedir. Yüksek sıkıştırma oranlarında JPEG-2000'in JPEG XR ile benzer performans gösterdiği ama JPEG XR'nin çok az da olsa üzerinde olduğu görülmektedir. Görece daha düşük sıkıştırma oranlarında JPEG XR'nin JPEG-2000'den daha iyi performans gösterdiği görülmektedir.



Şekil 4.3: Lenna görüntüsü PSNR ve SSIM sonuçları.

4.1.3.2 Barbara

Şekil 4.4'te Barbara görüntüsü için elde edilen PSNR ve SSIM sonuçları verilmiştir. PSNR/Bit derinliği grafiği incelendiğinde yüksek sıkıştırma oranlarında JPEG'in JPEG XR ve JPEG-2000'in gerisinde kaldığı görülmektedir. JPEG XR ile JPEG 2000 karşılaştırıldığında ise yüksek sıkıştırma oranlarında birbirine çok yakın performans gösterirken, JPEG-2000'in performansının az bir farkla da olsa JPEG XR'den iyi olduğu görülmektedir. Görece daha düşük sıkıştırma oranlarında ise JPEG-2000'in JPEG XR'nin gerisinde kaldığı görülmektedir.

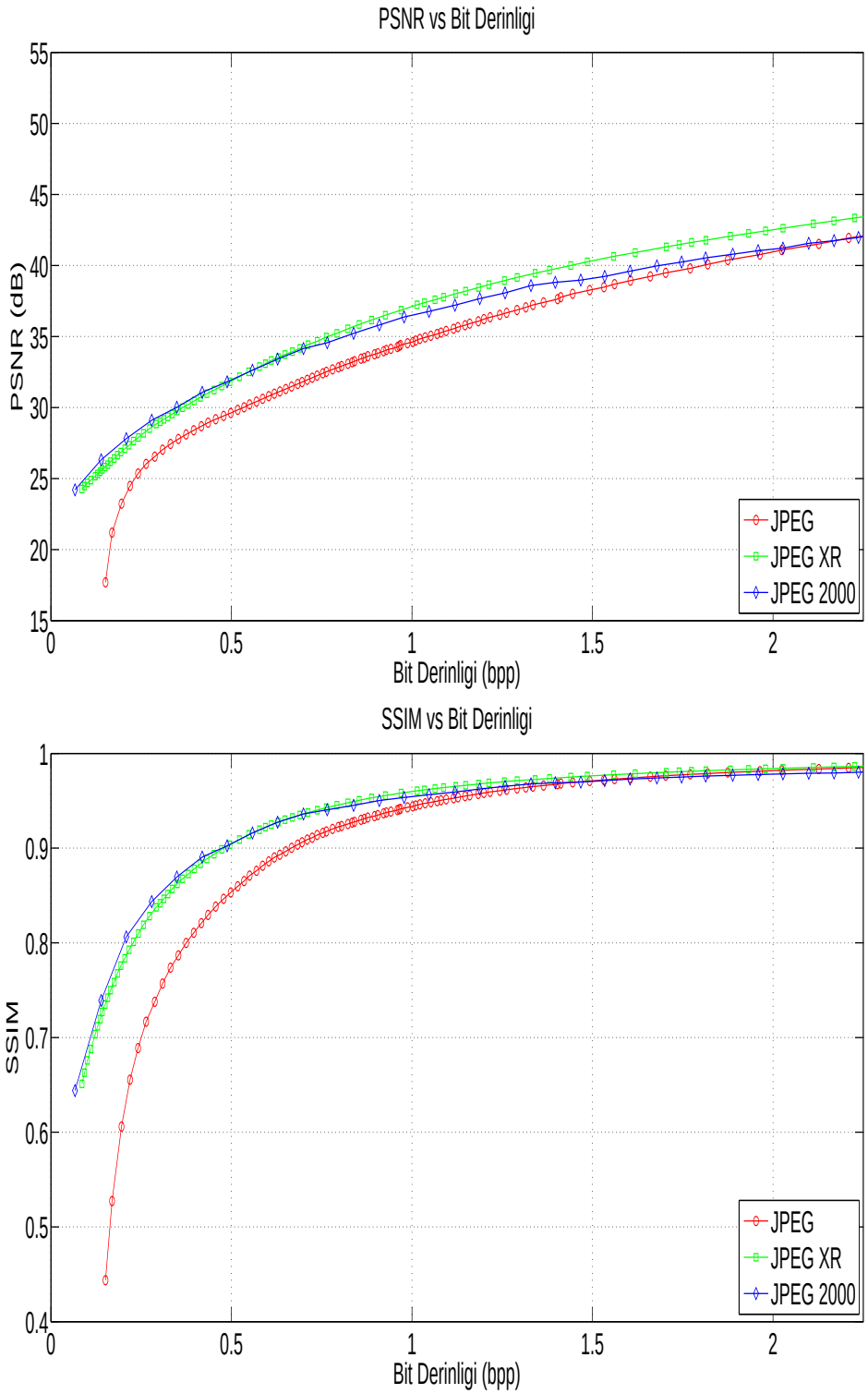
SSIM/Bit derinliği grafiği incelendiğinde yine yüksek sıkıştırma oranlarında JPEG XR ve JPEG-2000'in JPEG'e göre çok daha iyi performans gösterdiği görülmektedir. Yüksek sıkıştırma oranlarında JPEG-2000'in JPEG XR ile benzer performans gösterdiği ama JPEG-2000'in JPEG XR'nin çok az da olsa üzerinde olduğu görülmektedir. Görece daha düşük sıkıştırma oranlarında JPEG XR'nin JPEG-2000'den daha iyi performans gösterdiği görülmektedir.

4.1.3.3 Baboon

Baboon görüntüsü çok fazla detaya -tüy, kürk gibi- sahip olduğundan, renk geçişleri fazla olduğundan ve fazla miktarda doku bulundurduğundan sıkıştırılması zor olan bir görüntüdür. Şekil 4.5'te Baboon görüntüsü için elde edilen PSNR ve SSIM sonuçları verilmiştir. Grafikten de görüleceği üzere PSNR ve SSIM değerlerinde azalma söz konusudur. Yani aynı bit oranlarında Baboon görüntüsünün sıkıştırılmış hallerinin diğer test görüntülerine göre daha düşük PSNR ve SSIM değerlikleri verdiği görülmektedir. Bu da bu görüntü için sıkıştırma başarısının düştüğünü göstermektedir.

PSNR/Bit derinliği grafiği incelendiğinde yüksek sıkıştırma oranlarında JPEG'in JPEG XR ve JPEG-2000'in gerisinde kaldığı görülmektedir. JPEG XR ile JPEG 2000 karşılaştırıldığında ise yüksek sıkıştırma oranlarında birbirine çok yakın performans gösterdikleri görülmektedir. Görece daha düşük sıkıştırma oranlarında (yaklaşık 25 kattan daha az) ise JPEG-2000'in JPEG XR'nin gerisinde kaldığı görülmektedir.

SSIM/Bit derinliği grafiği incelendiğinde yine yüksek sıkıştırma oranlarında JPEG XR ve JPEG-2000'in JPEG'e göre çok daha iyi performans gösterdiği görülmektedir. Yüksek sıkıştırma oranlarında JPEG-2000'in JPEG XR ile benzer performans



Şekil 4.4: Barbara görüntüsü PSNR ve SSIM sonuçları.

gösterdiği hatta örtüştüğü görülmektedir. Görece daha düşük sıkıştırma oranlarında (yaklaşık 25 kattan daha az) JPEG XR'nin JPEG-2000'den daha iyi performans gösterdiği görülmektedir.

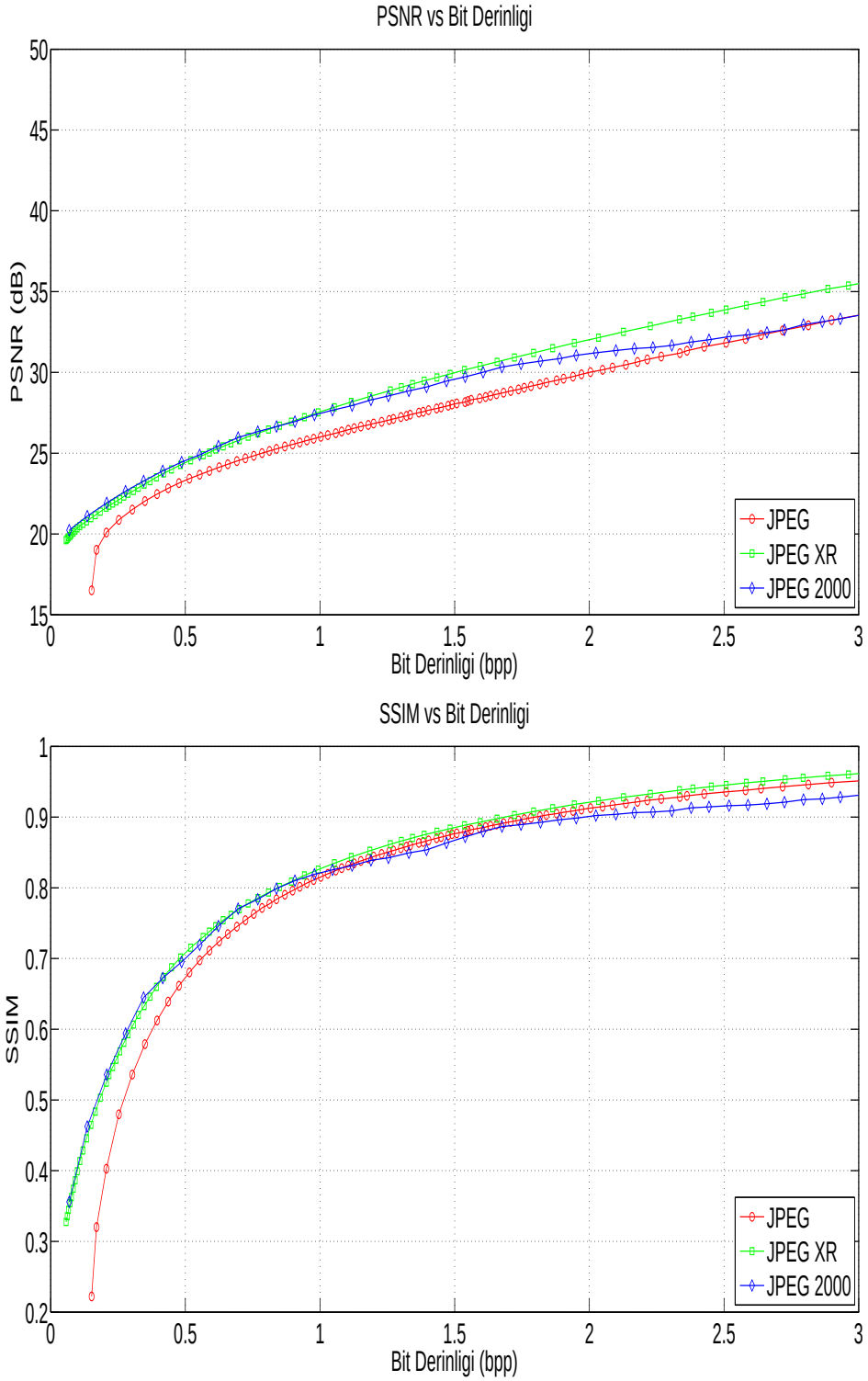
4.1.3.4 Lighthouse

Şekil 4.6'te Lighthouse görüntüsü için elde edilen PSNR ve SSIM sonuçları verilmiştir. PSNR/Bit derinliği grafiği incelendiğinde yüksek sıkıştırma oranlarında JPEG'in JPEG XR ve JPEG-2000'in gerisinde kaldığı görülmektedir. JPEG XR ile JPEG 2000 karşılaştırıldığında ise yüksek sıkıştırma oranlarında birbirine çok yakın performans gösterdikleri görülmektedir. Görece daha düşük sıkıştırma oranlarında (yaklaşık 15-16 kattan daha az) ise JPEG-2000'in JPEG XR'nin gerisinde kaldığı görülmektedir.

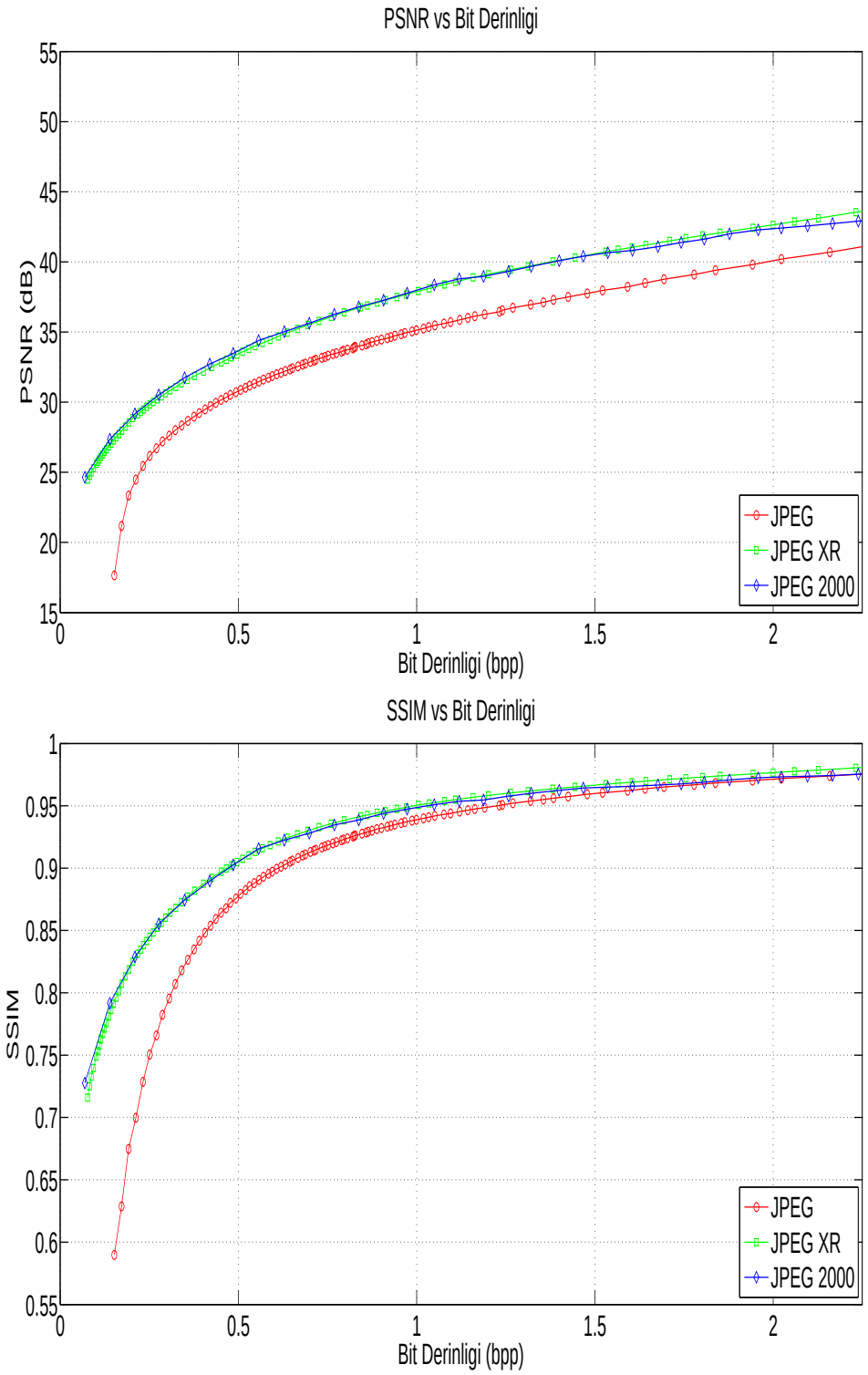
SSIM/Bit derinliği grafiği incelendiğinde yine yüksek sıkıştırma oranlarında JPEG XR ve JPEG-2000'in JPEG'e göre çok daha iyi performans gösterdiği görülmektedir. Yüksek sıkıştırma oranlarında JPEG-2000'in JPEG XR ile benzer performans gösterdiği hatta büyük oranda örtüştüğü görülmektedir. Görece daha düşük sıkıştırma oranlarında (yaklaşık 15-16 kattan daha az) JPEG XR'nin JPEG-2000'den daha iyi performans gösterdiği görülmektedir.

4.1.3.5 Ruler

Ruler görüntüsü yazı karakterleri ve çizgilerden oluşan sıkıştırma için zor bir görüntü türüdür. Şekil 4.7'te Ruler görüntüsü için elde edilen PSNR ve SSIM sonuçları verilmiştir. PSNR/Bit derinliği grafiği incelendiğinde yüksek sıkıştırma oranlarında elde edilen PSNR değerlerinin tüm kodeklerde oldukça düştüğü görülmektedir. JPEG-2000'in, JPEG XR'ye ve JPEG'e göre daha iyi bir performans sergilediği görülmektedir. JPEG XR'nin performansı zaman zaman JPEG'ten daha iyi olsa da JPEG'e oldukça yakındır. JPEG XR'nin bu görüntü için kararlı bir performans göstermediği görülmektedir. Burada dikkat edilmesi gereken noktalardan biri de PSNR/Bit derinliği grafiğinde JPEG-2000'in 1 ile 1.2 bit derinliği aralığında yaptığı büyük PSNR sıçramasıdır. Bu durumu incelemek için Ruler görüntüsü JASPER kodeki yardımıyla kayıpsız kodlanmış ve 769 KB boyutundaki Ruler görüntüsünün 37 KB boyutuna düştüğü görülmüştür. Bu oran, kayıpsız bir kodlama için görece yüksek bir sıkıştırma oranı olarak kabul edilebilir. Çünkü yine aynı boyuta -769 KB- sahip



Şekil 4.5: Baboon görüntüsü PSNR ve SSIM sonuçları.



Şekil 4.6: Lighthouse görüntüsü PSNR ve SSIM sonuçları.

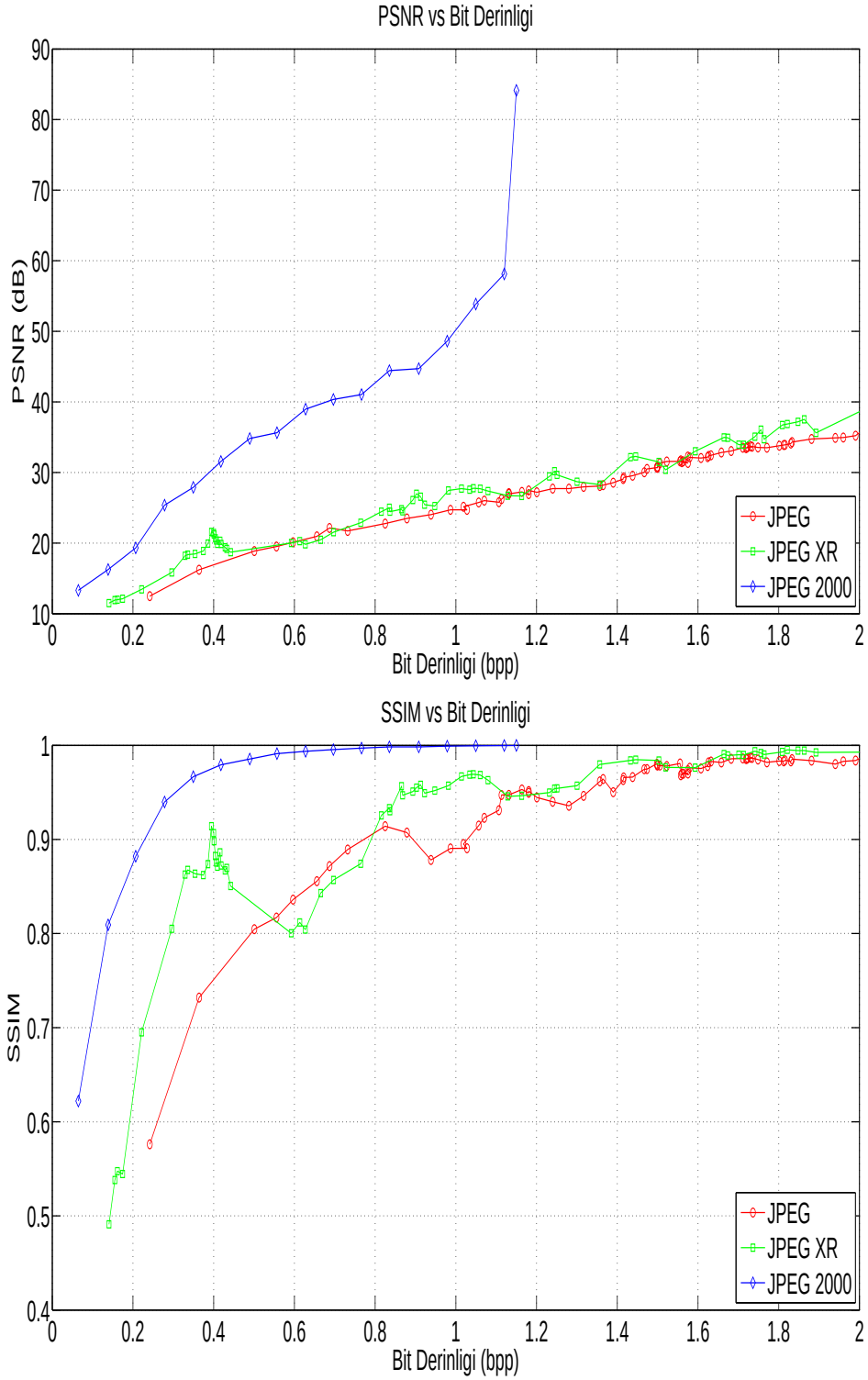
Lenna görüntüsü JASPER kodeki ile kayıpsız kodlandığında 435 KB boyutunda bir görüntü ortaya çıkmıştır. Bu durumla ilgili ilk akla gelen, Ruler görüntüsündeki bu ani PSNR sıçramasının sebebinin JASPER kodekinin bu noktadan itibaren kayıpsız kodlamaya geçmiş olması ihtimalidir. Bunu anlamanın yolu ise basittir. Yapılması gereken kayıpsız olarak JPEG-2000 kodlanmış Ruler görüntüsünün bit derinliğini hesaplamaktır. Bunun için JPEG-2000 kodlanmış Ruler görüntüsünün boyutu (37,708 bayt), orijinal görüntünün boyutuna (786,486 bayt) oranlanır ve elde edilen değer orijinal görüntünün bit derinliği (24 bit) ile çarpılır. Sonuç yaklaşık olarak 1.15 bpp çıkar ve bu da grafikteki sıçramayı açıklamaktadır. Tam bu noktada JASPER kodeki kayıpsız kodlamaya geçmiştir, bu yüzden PSNR değerinde de büyük bir sıçrama meydana gelmiştir.

SSIM/Bit derinliği grafiği incelendiğinde yine yüksek sıkıştırma oranlarında JPEG-2000'in diğer iki kodeki de geride bıraktığı görülmektedir. Yüksek sıkıştırma oranlarında JPEG XR'nin JPEG'ten daha iyi performans gösterdiği, fakat 20 kat ve daha az sıkıştırma oranlarında yine birbirine benzer performans sergiledikleri görülmektedir. JPEG XR'nin bu görüntü için kararlı bir performans sağlamadığı SSIM/Bit derinliği grafiğinde de görülmektedir.

Ruler görüntüsünde JPEG-2000'in JPEG XR ve JPEG'e üstünlük sağlamasının ana nedeni olarak ise bu standartlarda kullanılan dönüşüm türlerinin farklılığını gösterebiliriz. JPEG-2000 dalgacık (wavelet) dönüşümü kullanırken, JPEG XR ve JPEG frekans dönüşümleri kullanmaktadır.

4.2 Görsel Karşılaştırma ve Sonuçları

Görsel karşılaştırma, farklı görsel ve yapısal özelliklere sahip test görüntülerinin JPEG XR, JPEG ve JPEG-2000 kodekleri ile farklı kalite seviyelerinde kodlanmasını ve birbirine en yakın bit oranlarına sahip görüntülerin görsel olarak karşılaştırılmasını ifade etmektedir. Görsel karşılaştırma için yaklaşık 100 kat sıkıştırma yapılmıştır. JASPER kodlayıcı kodlama esnasında sıkıştırma oranını belirleyerek sıkıştırma yapma imkanı sağladığı için ilk olarak test görüntüsü JASPER ile kodlanmıştır. Elde edilen JASPER kodlanmış görüntünün dosya boyutlarına en yakın JPEG ve JPEG XR kodlanmış görüntüler alınarak görsel olarak karşılaştırılması amaçlanmıştır ve çizelgeler



Şekil 4.7: Ruler görüntüsü PSNR ve SSIM sonuçları.

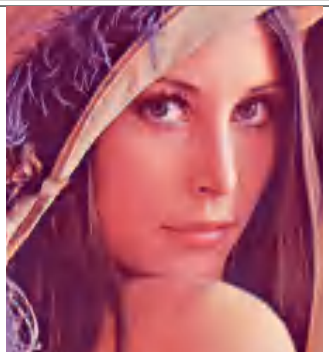
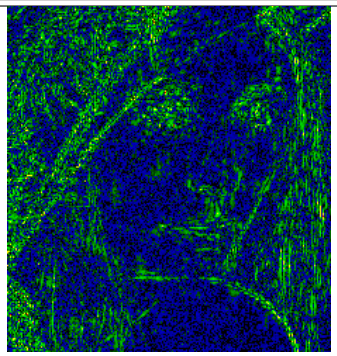
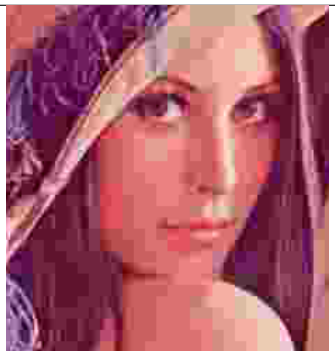
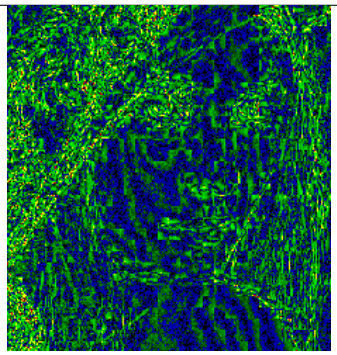
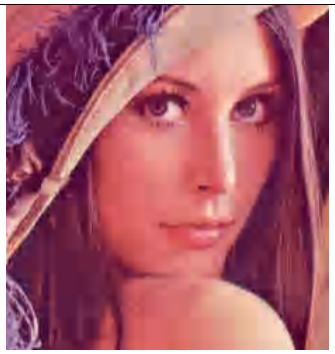
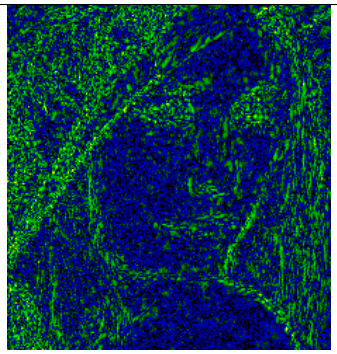
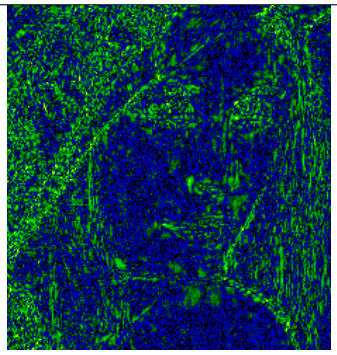
oluşturulmuştur. Bu çizelgeler oluşturulurken yukarıdan aşağı doğru JPEG-2000, JPEG ve JPEG XR kodlanmış görüntüler ve PSNR haritaları kullanılmıştır. Çizelge içerisindeki görüntülerin altlarında, kullanılan kodekler ve sıkıştırmada kullanılan parametreler belirtilmiştir. Aynı zamanda MSU Quality Measurement Tool Version 3.0 [19] ile elde edilmiş SSIM-YYUV ve PSNR-YYUV değerleri de çizelgelerde verilmiştir. SSIM-YYUV ve PSNR-YYUV değerlerindeki YYUV, bu metriklerin YUV renk uzayında Y bileşeni için hesaplandığını göstermektedir. Çizelgelerde verilen PSNR haritaları kodlanmış görüntülerin hata miktarları hakkında fikir vermektedir. Bu haritalarda kalite iyiden kötüye doğru şu renk sıralamasıyla belirlenmektedir: siyah, mavi, yeşil, sarı ve kırmızı. Görüntüler tamamen değil kısmi olarak çizelgelere eklenmiştir.

Lenna görüntüsünün görsel karşılaştırması Çizelge 4.5'te verilmiştir. Aradaki farkların daha belirgin anlaşılması için görsel karşılaştırmada Lenna görüntüsünün (157px, 193px) kordinatından itibaren (230px, 246px) genişliğine ve yüksekliğine sahip bir kısmı kullanılmıştır.

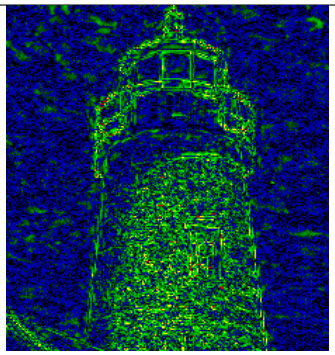
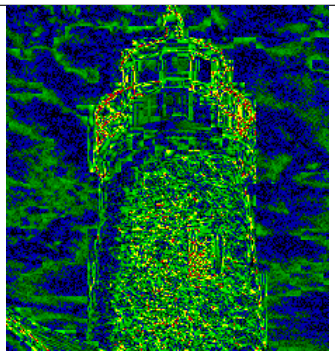
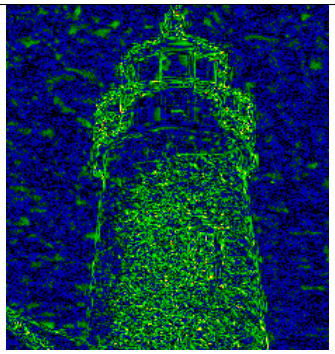
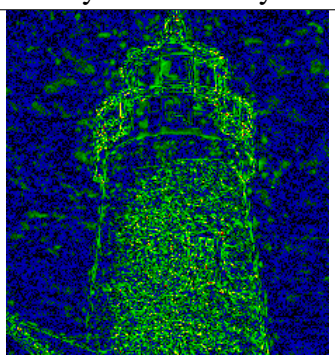
Çizelge 4.5'te de görüleceği gibi yaklaşık 100 kat sıkıştırma oranlarında en kötü görsel performansa sahip kodek JPEG'dir. Farklı "-l" parametrelerine (1, 2) göre JPEG XR kodlanmış görüntülerin "-l" parametresi 1 olarak kodlanmış görüntüde az da olsa bir bloklanma etkisi hissedilmektedir. Fakat kalite metriklerine bakıldığında en iyi performansı gösteren görüntü de yine bu görüntüdür. Bu görüntüyü JPEG-2000 kodlanmış görüntü ile karşılaştırdığımızda ise JPEG-2000'de bulanıklaşmadan ötürü detay kaybı olduğunu Lenna'nın şapkasındaki tüylü kısma bakarak anlayabiliriz. Bu görüşü desteklemek için aynı kalite seviyesinde "-l" parametresini 2 yaparak JPEG XR kodladığımızda görüntüdeki bulanıklaşmanın arttığı ve JPEG-2000 ile çok benzer bir sonuç yakalandığı görülmektedir. Kalite metrikleri bile neredeyse aynı değerlerdedir.

Lighthouse görüntüsünün görsel karşılaştırması Çizelge 4.6'te verilmiştir. Aradaki farkların daha belirgin anlaşılması için görsel karşılaştırmada Lighthouse görüntüsünün (190px, 105px) kordinatından itibaren (230px, 246px) genişliğine ve yüksekliğine sahip bir kısmı kullanılmıştır.

Çizelge 4.5: Lenna görüntüsü görsel karşılaştırma.

	
SSIM-YYUV = 0.8793 JP2: rate = 0.01	PSNR-YYUV = 33.0132 boyut = 7854 bayt
	
SSIM-YYUV = 0.7960 JPEG: quality = 6	PSNR-YYUV = 29.6363 boyut = 7725 bayt
	
SSIM-YYUV = 0.8803 JPEG XR: q = 72, l = 1	PSNR-YYUV = 33.2165 boyut = 7895 bayt
	
SSIM-YYUV = 0.8791 JPEG XR: q = 72, l = 2	PSNR-YYUV = 33.0013 boyut = 7698 bayt

Çizelge 4.6: Lighthouse görüntüsü görsel karşılaştırma.

	
SSIM-YYUV = 0.8614 JP2: rate = 0.01	PSNR-YYUV = 31.0617 boyut = 9768 bayt
	
SSIM-YYUV = 0.7584 JPEG: quality = 5	PSNR-YYUV = 26.7665 boyut = 9517 bayt
	
SSIM-YYUV = 0.8628 JPEG XR: q = 77, l = 1	PSNR-YYUV = 30.9934 boyut = 9934 bayt
	
SSIM-YYUV = 0.8628 JPEG XR: q = 77, l = 2	PSNR-YYUV = 30.7939 boyut = 9799 bayt

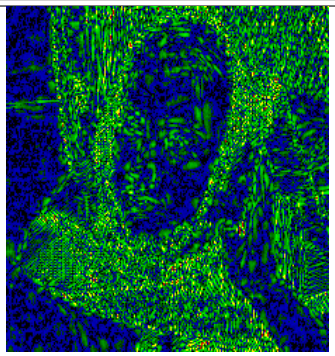
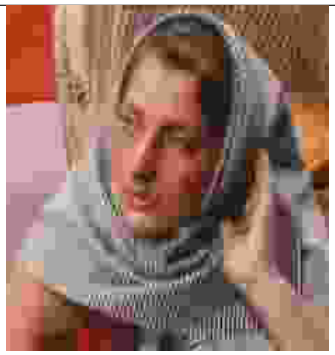
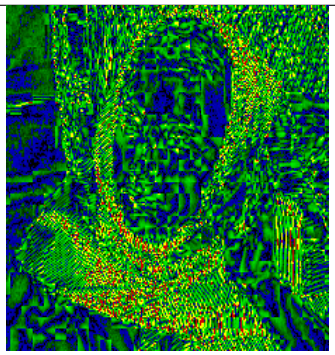
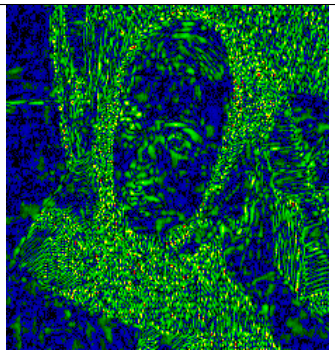
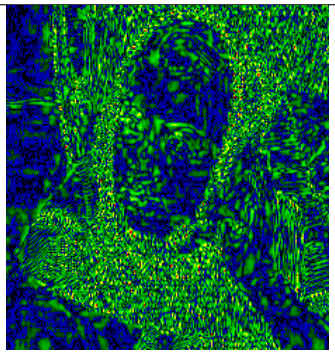
Çizelge 4.6'ya bakıldığında yaklaşık 100 kat sıkıştırma oranlarında Lighthouse görüntüsü için JPEG performansının görsel olarak oldukça kötü durumda olduğu görülmektedir. JPEG XR kodlanmış görüntülerden "-l" parametresi 1 olarak kodlanan görüntüde bulutlarda bloklanma etkisi gözlemlenirken detayların JPEG-2000 kodlanmış görüntüye göre daha belirgin olduğu görülmektedir. Fenerin taş duvarı incelendiğinde JPEG-2000 kodlanmış görüntüde oldukça detay kaybı olduğu ve çok fazla bulanıklaşma etkisi görülmektedir. JPEG-2000 kodlanmış görüntü ile JPEG XR kodlanmış görüntülerin PSNR haritaları karşılaştırıldığında da bu gayet açık bir şekilde görülmektedir. Çünkü JPEG-2000 kodlanmış görüntünün PSNR haritasına bakıldığında fenerin taş duvarları olan kısımda kırmızı ve sarı piksellerin JPEG XR kodlanmış görüntülere göre daha fazla olduğu görülmektedir. Bu piksellerin daha fazla olması, görüntünün o kısmındaki piksellerin orijinal görüntüdekilerle olan benzerliğinin azaldığına işaret etmektedir.

Barbara görüntüsünün görsel karşılaştırması Çizelge 4.7'de verilmiştir. Aradaki farkların daha belirgin anlaşılması için görsel karşılaştırmada Barbara görüntüsünün (325px, 42px) kordinatından itibaren (230px, 246px) genişliğine ve yüksekliğine sahip bir kısmı kullanılmıştır.

Çizelge 4.7'ye bakıldığında yaklaşık 100 kat sıkıştırma oranlarında Barbara görüntüsü için JPEG performansının yine görsel olarak oldukça kötü sonuç verdiği görülmektedir. JPEG XR kodlanmış görüntülerden "-l" parametresi 1 olarak kodlanan görüntüde Barbara'nın yanak bölgesindeki bloklanma efektleri kolayca görülmektedir. "l" parametresi 2 olarak JPEG XR kodlanan görüntüde ise bulanıklaşma ve detay kaybı görülmektedir. Barbara'nın sol gözüne bakıldığında bu durum daha iyi anlaşılmaktadır. JPEG-2000 kodlamanın bu görüntü için en iyi sonucu verdiği söylenebilir ama JPEG XR kodlanmış görüntüler ile karşılaştırıldığında çok büyük bir fark görülmemektedir. PSNR ve SSIM metrikleri de görsel karşılaştırmayı destekler niteliktedir. JPEG-2000 ve JPEG XR kodlanmış görüntüler için bu değerler açısından büyük farklar görülmemektedir.

JPEG XR'nin JPEG ve JPEG-2000 ile öznel karşılaştırmasını ve bu karşılaştırmanın detaylı bir istatistiksel analizini içeren bir de çalışma bulunmaktadır [22]. Bu çalışma,

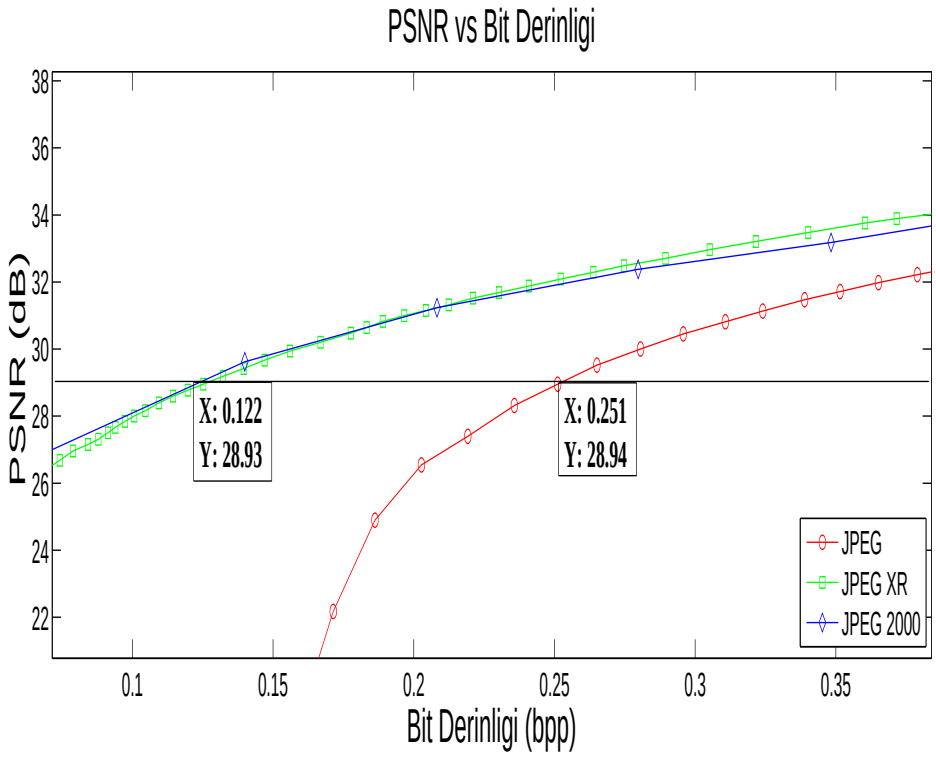
Çizelge 4.7: Barbara görüntüsü görsel karşılaştırma.

	
SSIM-YYUV = 0.8384 JP2: rate = 0.01	PSNR-YYUV = 29.6180 boyut = 9820 bayt
	
SSIM-YYUV = 0.7176 JPEG: quality = 5	PSNR-YYUV = 26.6754 boyut = 9889 bayt
	
SSIM-YYUV = 0.8299 JPEG XR: q = 83, l = 1	PSNR-YYUV = 29.2165 boyut = 9933 bayt
	
SSIM-YYUV = 0.8268 JPEG XR: q = 83, l = 2	PSNR-YYUV = 29.0263 boyut = 9556 bayt

bit oranları ile sıkıştırılmış görüntülerdeki algılanan kalitenin ilişkisini de ortaya koymaktadır.

4.3 Genel Sonuçlar ve Yorumlar

JPEG XR kodlama hem kalite metrikleri açısından hem de görsel açıdan JPEG kodlamadan üstün bir performans sergilemiştir. Yüksek sıkıştırma oranlarında Microsoft'un iddia ettiği gibi JPEG XR, JPEG'in 2 katından daha iyi bir sıkıştırma kalitesine sahiptir [23]. Burada sıkıştırma kalitesi ile belirtilmek istenen, referans bir kalite seviyesi için kodlanmış dosyaların dosya boyutlarının ne derece küçük olduğudur. Örneğin aynı PSNR değerine sahip JPEG ve JPEG XR kodlanmış iki görüntünün dosya boyutları karşılaştırıldığında JPEG XR'nin dosya boyutu daha küçük olduğundan sıkıştırma kalitesi daha yüksektir. Bunun için daha önce de elde edilmiş olan Lenna görüntüsünün PSNR sonuçlarını içeren grafiğini ya da ImageQualityClass'a ait FindClosestPSNRs() metodunu kullanabiliriz. Bu metod JPEG ve JPEG XR kodlanmış görüntülerin PSNR değerlerini birbiriyle karşılaştırarak birbirine en yakın PSNR'a sahip olan görüntüleri ve bu PSNR değerlerini indekslemektedir. Şekil 4.8'de Lenna görüntüsünün PSNR sonuçları ve yaklaşık 28.95 dB'deki bit derinlikleri bir veri imleci vasıtasıyla verilmiştir. Burada görüldüğü gibi yaklaşık olarak 29dB civarında JPEG XR kodlanmış görüntünün bit derinliği 0.122 bpp, JPEG'inki ise 0.251 bpp'dir. 0.251 değeri 0.122'ye bölündüğünde yaklaşık sonuç olarak 2.06 elde edilir. Yani yaklaşık 29dB PSNR seviyesinde JPEG kodlanmış görüntünün dosya boyutu JPEG XR'ninkinin 2.06 katıdır.Bu seviyeden itibaren daha yüksek sıkıştırma oranlarında PSNR düşecek fakat aradaki makas daha da açılarak 2.06 değeri büyüyecektir. ImageQualityClass nesnesine ait özellikler -PSNR değerleri ve kalite seviyeleri- incelendiğinde de JPEG XR için 86 kalite seviyesinde kodlanmış, PSNR değeri 28.95 dB ve dosya boyutu 4102 bayt olan görüntü ile JPEG için 7 kalite seviyesinde kodlanmış, PSNR değeri 28.95 dB ve dosya boyutu 8229 bayt olan görüntülerin PSNR değerlerinin aynı olduğu görülebilir. Dosya boyutları oranlandığında yaklaşık 2.006 değeri elde edilir. Bu sonuçtan da yüksek sıkıştırma oranlarında, belirli bir kalite seviyesinden itibaren, JPEG XR'nin JPEG'in iki katından daha iyi bir sıkıştırma kalitesine sahip olduğu söylenebilir.



Şekil 4.8: Lenna görüntüsünün PSNR sonuçlarında 28.95 dB civarı bit derinlikleri.

Dikkat edilmesi gereken nokta bu iddianın geçerli olduğu kalite seviyesinin kabul edilir bir seviye olup olmamasıdır ki bu deneyde kalite seviyesi 29 dB civarındaki PSNR değeridir. Yani bu kalite seviyesinin üstünde Microsoft'un iddiası geçerliliğini yitirirken, altında ise geçerliliğini sürdürmektedir.

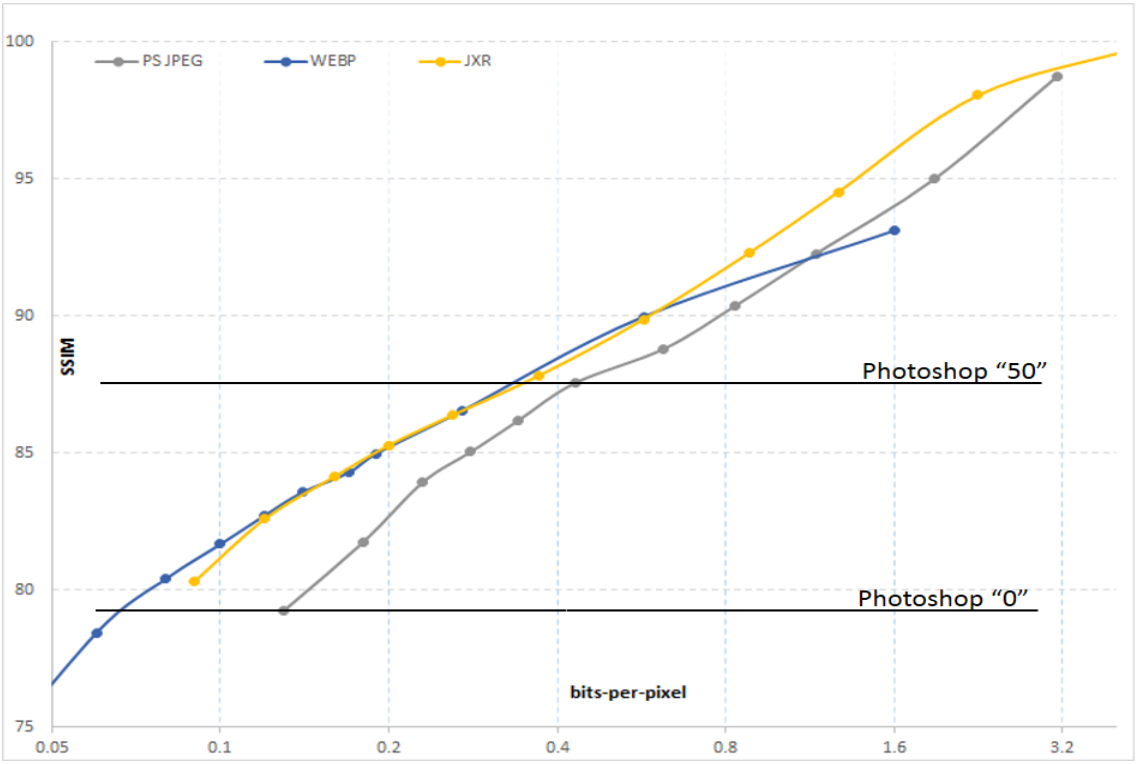
JPEG XR ve JPEG-2000'in nesnel karşılaştırması sonucunda yüksek sıkıştırma oranlarında benzer peformans gösterdikleri; fakat görece düşük sıkıştırma oranlarında JPEG XR'nin JPEG-2000'i geçtiği görülmektedir. JPEG XR JPEG-2000 ile karşılaştırıldığında görüntü sıkıştırmada bir devrim niteliği taşımamaktadır. Microsoft da JPEG XR'nin JPEG-2000'e benzer performans sergilediğini deklare etmiştir [23].

JPEG XR ve JPEG-2000 kodlamaların görsel karşılaştırması sonucunda birbirlerinden büyük bir farkları olmadığı, görüntüden görüntüye değişen avantaj ve dezavantajları olduğu görülmektedir.

Şimdiye kadar JPEG-2000'in kullanım alanının sınırlı kalması (tarayıcılarda, resim editörlerinde bulunmaması) JPEG XR için bir avantaj olup, artan görüntü sıkıştırma ihtiyacı için JPEG XR'nin bir çıkış yolu olmasını sağlayabilir. JPEG XR'nin işlemsel karmaşıklığı JPEG-2000'den daha az olduğu için, yüksek çözünürlüklü görüntülerin

sıkıştırılmasında benzer bit oranları ve PSNR değerleri için hem yazılımsal olarak hem de donanımsal olarak JPEG XR kullanmak avantajlı olacaktır.

JPEG'in yerini alabileceği düşünelerek Google tarafından ortaya atılan WebP görüntü formatı ile JPEG XR'nin karşılaştırılmasına ilişkin bir çalışma bulunmaktadır [4]. Bu çalışmada Google'nin kendi çalışmalarında kullandığı test görüntüleri kullanılmış; kodlamalar esnasında JPEG XR, WebP ve Photoshop'un JPEG kodlayıcısı kullanılmıştır. Her kodlayıcı için 10 farklı kalite seviyesinde kodlama yapılmış ve Lenna görüntüsü için elde edilen sonuçlar Şekil 4.9'da verilmiştir. Şekil



Şekil 4.9: Lenna görüntüsüne ait WebP, JPEG XR ve Photoshop JPEG için SSIM sonuçları [4].

4.9'da görüldüğü gibi görece yüksek sıkıştırma oranlarında WebP ile JPEG XR performansları çok yakınken, daha düşük sıkıştırma oranlarında JPEG XR WebP'ye göre daha iyi performans göstermiştir. Grafikteki yatay Photoshop çizgileri 0-100 ölçeğinde 0 ve 50 kalite seviyelerini gösteren bir belirteç olarak kullanılmıştır. Yakın zamanda JPEG XR ve WebP karşılaştırmasına bir de HEVC-MSP eklenerek Mozilla tarafından yeni bir çalışma daha yapılmıştır [24]. Fakat bu çalışmada minimum kalite seviyesi için JPEG'in 50 kalite seviyesi baz alındığı için görece düşük sıkıştırma oranlarında gerçeklenmiş bir çalışma olduğu söylenebilir. Daha objektif karşılaştırma

için bu çalışma yüksek sıkıştırma oranları için de bir gelecek çalışması olarak yapılabilir.

KAYNAKLAR

[1] **Saba, B.**, 2012. JPEG XR scalable coding for remote image browsing applications, *Doktora Tezi*.

[2] **ISO/IEC 29199-2**, 2010, Information technology –JPEG XR image coding system – Part 2: Image coding specification.

[3] **Lang Yu**, 2010. Evaluating and implementing JPEG XR optimized for video surveillance, *Yüksek Lisans Tezi*.

[4] **Uyttendaele, M.**, 2013, JPEGXR Updates, `http://hdview.wordpress.com/2013/05/30/jpegxr-updates/`, alındığı tarih: 01.12.2013.

[5] **ITU-T Recommendation T.81 | ISO/IEC 10918-1**, 1992, Information technology - Digital compression and coding continuous-tone still images - Part1: Requirements and guidelines.

[6] **ISO/IEC 15444-1**, 2000, Information technology - JPEG 2000 image coding system - Part 1: Core coding system.

[7] **ISO/IEC 29199-1**, 2010, Information technology – JPEG XR image coding system – Part 1: System architecture.

[8] **Dufaux, F.**, **Sullivan, G.J. ve Ebrahimi, T.**, 2009. The JPEG XR Image Coding Standard, *IEEE Signal Processing Magazine*, **26(6)**, 195–199,204–204.

[9] **Douglas A. Kerr**, 2005, Chrominance Subsampling in Digital Images, `http://dougkerr.net/pumpkin/articles/Subsampling.pdf`.

[10] **Ahmed, N.**, **Natarajan, T. ve Rao, K.R.**, 1974. Discrete Cosine Transform, *IEEE Transactions on Computers*, **23(1)**, 90–93.

[11] **Jadhav, S.S. ve Jadhav, S.K.**, 2012. JPEG XR an Image Coding Standard, *Internationl Journal of Computer and Electrical Engineering*, **4(2)**, 137–140.

[12] **ISO/IEC 29199-3**, 2010, Information technology –JPEG XR image coding system – Part 3: Motion JPEG XR.

[13] **ISO/IEC 29199-4**, 2010, Information technology –JPEG XR image coding system – Part 4: Conformance testing.

[14] **ISO/IEC 29199-5**, 2010, Information technology –JPEG XR image coding system – Part 5: Reference software.

[15] **Crow, B.**, 2009, JPEG XR is Now an International Standard, http://blogs.msdn.com/b/billcrow/archive/2007/03.aspx, alındığı tarih: 07.11.2013.

[16] **Jpeg (Ver. 9a)**, Bilgisayar Yazılımı, http://www.ijg.org/, alındığı tarih: 15.09.2013.

[17] **Jasper (Ver. 1.900.1)**, Bilgisayar Yazılımı, http://www.ece.uvic.ca/~frodo/jasper/, alındığı tarih: 15.09.2013.

[18] **IrfanView (Ver. 4.36)**, Bilgisayar Yazılımı, http://www.irfanview.com/, alındığı tarih: 15.09.2013.

[19] **MSU Video Quality Measurement Tool (Ver. 3.0)**, Bilgisayar Yazılımı, http://www.compression.ru/video/quality_measure/vqmt_download_en.html, alındığı tarih: 15.09.2013.

[20] **Wang, Z.**, **Bovik, A.C. ve Sheikh H. R., S.E.P.**, 2004. Image Quality Assessment: From Error Visibility to Structural Similarity, *IEEE Transactions on Image Processing*, **13(4)**, 600–612.

[21] **Wang, Z.**, **Bovik, A.C. ve Sheikh H. R., S.E.P.**, The SSIM Index for Image Quality Assessment, https://ece.uwaterloo.ca/~z70wang/research/ssim/, alındığı tarih: 1.09.2013.

[22] **Simone, F.D.**, **Goldmann, L.**, **Baroncini, V. ve Ebrahimi, T.**, 2009. Subjective Evaluation of JPEG XR Image Compression, *Proc. SPIE*, **7443**.

[23] **Microsoft Corp.**, 2010, HD Photo Specification Download, http://msdn.microsoft.com/en-us/windows/hardware/gg463400.aspx, alındığı tarih: 28.11.2013.

[24] **Aas, J.**, **Maxwell, G.**, **Muizelaar, J. ve Terriberry, T.**, 2013, Lossy Compressed Image Formats Study, http://people.mozilla.org/~josh/lossy_compressed_image_study_october_2013/, alındığı tarih: 01.12.2013.

EKLER

EK A.1 : JPEG sınıfı Matlab kodu
EK A.2 : JPEG XR sınıfı Matlab kodu
EK A.3 : JPEG 2000 sınıfı Matlab kodu
EK A.4 : ImageQuality sınıfı Matlab kodu
EK A.5 : JPEG XR ITU-T Rec. T.835 Referans yazılımı; JASPER-1.900.1 yazılımı; Jpeg-9 yazılımı; PSNR ve SSIM Matlab kodları; JPEG, JPEG XR, JPEG 2000 ve ImageQuality Matlab sınıf kodları; test görüntüleri (CD)

EK A.1

```
classdef JpegClass < handle
    %   Author : Sezer BAGLAN
    %   Date: 17.09.2013
    %   JpegClass uses standard jpeg-9 encoder and decoder executables.
    %   The aim of this class is to do necessary encoding and decoding
    %   operations before comparison stage of my thesis.

    properties
        cjpeg               %standard cjpeg.exe (encoder)
        djpeg               %standard djpeg.exe (decoder)
        inputFile           %input file for encode-decode
        outputFile          %output file for encode-decode
        encoderArgs         %Encoder Commandline arguments
        decoderArgs         %Decoder Commandline arguments
        min_quality = 0;    %minimum quality metric for JPEG encoder
        max_quality = 98;  %maximum quality metric for JPEG encoder.100 by default, NA.
        Nstep               %step number between min-max quality vals.
    end


    methods
        function JPEGobj = JpegClass(cjpeg, djpeg, inputFile, outputFile, NStep)
            JPEGobj.cjpeg = cjpeg;
            JPEGobj.djpeg = djpeg;
            JPEGobj.inputFile = inputFile;
            JPEGobj.outputFile = outputFile;
            JPEGobj.Nstep = NStep;
        end

        %cjpeg.exe [arguments] inputfile outputfile
        function doJpegSimpleEncode(JPEGobj, encoderArgs, inputFile, outputFile)
            finalCommand = [JPEGobj.cjpeg ' ' encoderArgs ' ' inputFile ' ' outputFile];
            system([finalCommand]);
        end

        %Gets encoder and decoder arguments.
        %According to Nstep size firstly encodes the input image.
        %After encoding process, decodes each encoded image.
        function doJpegEncodeDecodeTest(JPEGobj, encoderArgs, decoderArgs)
            quality = 0;
            jpegTitle ='.jpeg';
            bmpTitle ='.bmp';
            q_arg = '-q';
            JPEGobj.encoderArgs = encoderArgs;
            JPEGobj.decoderArgs = decoderArgs;
            steps = ((JPEGobj.max_quality -JPEGobj.min_quality) / JPEGobj.Nstep);
            for i=1:JPEGobj.Nstep;
                quality = quality + steps;
                quality_str = int2str(quality);
                %encode%
                finalCommandEncode = [JPEGobj.cjpeg ' ' q_arg ' ' quality_str ' ' JPEGobj.encoderArgs
                ' ' JPEGobj.inputFile bmpTitle ' ' JPEGobj.outputFile quality_str jpegTitle];
                system([finalCommandEncode]);
                %decode: Default decoding: bmp%
                finalCommandDecode = [JPEGobj.djpeg ' ' JPEGobj.decoderArgs ' ' JPEGobj.outputFile
                quality_str jpegTitle ' ' JPEGobj.outputFile quality_str bmpTitle];
                system([finalCommandDecode]);
            end % for i ..
        end

        %djpeg.exe [arguments] inputfile outputfile
        function doJpegSimpleDecode(JPEGobj, cmd_arguments, inputFile, outputFile)
            finalCommand = [JPEGobj.djpeg ' ' cmd_arguments ' ' inputFile ' ' outputFile];
```

```
            system([finalCommand]);
        end


        function printEncodeOptions(JPEGobj)
            finalCommand = [JPEGobj.cjpeg];
            system([finalCommand]);
        end

        function printDecodeOptions(JPEGobj)
            finalCommand = [JPEGobj.djpeg];
            system([finalCommand]);
        end

    end

end
```

EK A.2

```
classdef JpegXRClass < handle
    %   Author : Sezer BAGLAN
    %   Date: 17.09.2013
    %   JpegXRClass uses standard JXR_T-REC-T.835-201001 encode/decode
    %   The aim of this class is to do necessary encoding and decoding
    %   operations before comparison stage of my thesis.

    properties
    jpegxr              %encoder and decoder
    inputFile           %input file for encode-decode
    outputFile          %output file for encode-decod
    encoderArgs         %Encoder Commandline arguments
    decoderArgs         %Decoder Commandline arguments
    min_quality = 0;    %minimum quality metric for JpegXR encoder
    max_quality = 100;  %maximum quality metric for JpegXR encoder. 255 by default, NA.
    Nstep               %step number between min-max quality vals.
    BmpToTifConv        %Use IrfanView for conversion
    end

    methods
        %Give inputFile without the file extension!!!e.g lena.bmp->lena
        function JpegXRobj = JpegXRClass(jpegxr, inputFile, outputFile, NStep, BmpToTifConv)
            JpegXRobj.jpegxr = jpegxr;
            JpegXRobj.inputFile = inputFile;
            JpegXRobj.outputFile = outputFile;
            JpegXRobj.Nstep = NStep;
            JpegXRobj.BmpToTifConv = BmpToTifConv;
        end

        function doBMPtoTIFconversion(JpegXRobj, inputFile, outputFile)
            convCommand = [JpegXRobj.BmpToTifConv ' ' inputFile ' ' '/convert=' outputFile];
            system([convCommand]);
        end

        %jpegxr.exe [arguments] inputfile
        function doJpegXRSimpleEncodeOrDecode(JpegXRobj, encoderArgs, inputFile)
            finalCommand = [JpegXRobj.jpegxr ' ' encoderArgs ' ' inputFile];
            system([finalCommand]);
        end

        %Gets encoder and decoder arguments.
        %According to Nstep size firstly encodes the input image.
        %After encoding process, decodes each encoded image.
        function doJpegXREncodeDecodeTest(JpegXRobj, encoderArgs, decoderArgs)
            quality = 0;
            bmpTitle ='.bmp';
            jxrTitle = '.jxr';
            tifTitle = '.tif';

            in = [JpegXRobj.inputFile bmpTitle];
            out = [JpegXRobj.inputFile tifTitle];
            JpegXRobj.doBMPtoTIFconversion(in, out);

            jxrEncodeOptions = '-c -o';
            jxrDecodeOptions = '-o';
            jxrFileName = JpegXRobj.outputFile;%'Test_JXR_q_';
            q_arg = '-q';

            JpegXRobj.encoderArgs = encoderArgs;
            JpegXRobj.decoderArgs = decoderArgs;
            steps = ((JpegXRobj.max_quality -JpegXRobj.min_quality) / JpegXRobj.Nstep);
            for i=1:JpegXRobj.Nstep;
                quality = quality + steps;
                quality_str = int2str(quality);
                %encode%
                finalCommandEncode = [JpegXRobj.jpegxr ' ' jxrEncodeOptions ' ' jxrFileName
                quality_str jxrTitle ' ' q_arg ' ' quality_str ' ' JpegXRobj.encoderArgs ' ' out];
```

```
                system([finalCommandEncode]);
                %decode: Default decoding: bmp%
                finalCommandDecode = [JpegXRobj.jpegxr ' ' jxrDecodeOptions ' ' jxrFileName
                quality_str tifTitle ' ' jxrFileName quality_str jxrTitle];
                system([finalCommandDecode]);
                %convert TIF files to BMP
                inputConv = [jxrFileName quality_str tifTitle];
                outputConv = [jxrFileName quality_str bmpTitle];
                JpegXRobj.doBMPtoTIFconversion(inputConv, outputConv);
            end % for i ..
        end

        function printEncodeDecodeOptions(JpegXRobj)
            finalCommand = [JpegXRobj.jpegxr];
            system([finalCommand]);
        end


    end

end
```

EK A.3

```
classdef Jpeg2000Class < handle
    %   Author : Sezer BAGLAN
    %   Date: 26.09.2013
    %   Jpeg2000Class uses jasper-1.900.1 encoder/decoder executables.
    %   The aim of this class is to do necessary encoding and decoding
    %   operations before comparison stage of my thesis.

    properties
        jasper              %standard jasper.exe (encoder/decoder)
        inputFile           %input file for encode-decode
        outputFile          %output file for encode-decode
        encoderArgs         %Encoder Commandline arguments
        decoderArgs         %Decoder Commandline arguments
        min_quality = 0;    %minimum quality metric for JP2 encoder
        max_quality = 0.35; %maximum quality metric for JP2 encoder.
         %Max value:1 but set to 0,35 for comparison.
        Nstep               %step number between min-max quality vals.
    end


    methods
        function Jpeg2000obj = Jpeg2000Class(jasper, inputFile, outputFile, NStep)
            Jpeg2000obj.jasper = jasper;
            Jpeg2000obj.inputFile = inputFile;
            Jpeg2000obj.outputFile = outputFile;
            Jpeg2000obj.Nstep = NStep;
        end

        %jasper.exe -f input -F output -T jp2 -o rate=0.01
        function doJpeg2000SimpleEncode(Jpeg2000obj, encoderArgs, inputFile, outputFile)
            finalCommand = [Jpeg2000obj.jasper ' -f ' inputFile ' -F '
            outputFile ' -T ' encoderArgs];
            system([finalCommand]);
        end

        %Gets encoder and decoder arguments.
        %According to Nstep size firstly encodes the input image.
        %After encoding process, decodes each encoded image.
        function doJpeg2000EncodeDecodeTest(Jpeg2000obj, encoderArgs, decoderArgs)
            quality = 0;
            jp2Title ='.jp2';
            bmpTitle ='.bmp';
            Jpeg2000obj.encoderArgs = encoderArgs;
            Jpeg2000obj.decoderArgs = decoderArgs;
            steps = ((Jpeg2000obj.max_quality -Jpeg2000obj.min_quality) / Jpeg2000obj.Nstep);
            for i=1:Jpeg2000obj.Nstep;
                quality = quality + steps;
                quality_str = num2str(quality);
                %encode%
                finalCommandEncode = [Jpeg2000obj.jasper ' -f ' Jpeg2000obj.inputFile bmpTitle
                ' -F ' Jpeg2000obj.outputFile quality_str jp2Title ' -T jp2 -O rate='
                quality_str Jpeg2000obj.encoderArgs];
                system([finalCommandEncode]);
                %decode: Default decoding: bmp%
                finalCommandDecode = [Jpeg2000obj.jasper ' -f ' Jpeg2000obj.outputFile quality_str
                jp2Title ' -F ' Jpeg2000obj.outputFile quality_str bmpTitle ' -T bmp'];
                system([finalCommandDecode]);
            end % for i ..
        end


        function printEncodeDecodeOptions(Jpeg2000obj)
            finalCommand = [Jpeg2000obj.jasper ' --help'];
            system([finalCommand]);
        end
```

```
    end
end
```

EK A.4

```
classdef ImageQualityClass < handle
    %   Author : Sezer BAGLAN
    %   Date: 17.09.2013
    %   Aims to measure quality metrics PSNR and SSIM and do some
    %   comparison stuff related to my thessis.

    properties
        jpegObj             %JpegClass object
        jxrObj              %JpegXRClass object
        jp2Obj              %Jpeg2000Class object
        jpegPSNR            %holds PSNR of JPEG encoded images.
        jpegSSIM            %holds SSIM of JPEG encoded images.
        jxrPSNR             %holds PSNR of JPEG XR encoded images.
        jxrSSIM             %holds SSIM of JPEG XR encoded images.
        jp2PSNR             %holds PSNR of JPEG 2000 encoded images.
        jp2SSIM             %holds SSIM of JPEG 2000 encoded images.
        jpegSSIMmap         %holds SSIM Maps of JPEG encoded images.
        jxrSSIMmap          %holds SSIM Maps of JPEG XR encoded images.
        jp2SSIMmap          %holds SSIM Maps of JPEG 2000 encoded images.
        targetPSNRs         %holds closest PSNR values after comparison.
        targetSSIMs         %holds closest SSIM values after comparison.
        targetJpegId        %holds the image ids of compared jpeg images.
        targetJxrId         %holds the image ids of compared jxr images.
        jpegFileSizeVector  %holds file sizes of jpeg encoded images.
        jxrFileSizeVector   %holds file sizes of jxr encoded images.
        jp2FileSizeVector   %holds file sizes of jpeg 2000 encoded images.
        width               %holds the width of image
        height              %holds the height of image
        dimension           %holds the dimension of image. e.g. RGB = 3
    end

    methods
        %Constructor%
        function ImgQualobj = ImageQualityClass(jpegObj, jxrObj, jp2Obj)
            ImgQualobj.jpegObj = jpegObj;
            ImgQualobj.jxrObj = jxrObj;
            ImgQualobj.jp2Obj = jp2Obj;
        end

        %Calculate PSNR values of JPEG, JXR and JP2000 encoded images in workspace
        function CalculatePSNRs (ImgQualobj)
        bmpTitle = '.bmp';
        Iref = imread([ImgQualobj.jpegObj.inputFile bmpTitle]);
        imgSize = size(Iref);
        ImgQualobj.width = imgSize(1);
        ImgQualobj.height = imgSize(2);
        ImgQualobj.dimension = imgSize(3);
        quality = 0;
        Iref = rgb2gray(Iref);
        steps = ((ImgQualobj.jpegObj.max_quality -ImgQualobj.jpegObj.min_quality) /
        ImgQualobj.jpegObj.Nstep);
        ImgQualobj.jpegPSNR = zeros(1,ImgQualobj.jpegObj.Nstep);
            for i=1:ImgQualobj.jpegObj.Nstep;
                quality = quality + steps;
                quality_str = int2str(quality);
                jpegFile = [ImgQualobj.jpegObj.outputFile quality_str bmpTitle];
                I = rgb2gray(imread(jpegFile));
                ImgQualobj.jpegPSNR(1,i) = psnr(double(I),double(Iref));
            end % for i ..

        quality = 0;
        steps = ((ImgQualobj.jxrObj.max_quality -ImgQualobj.jxrObj.min_quality) /
        ImgQualobj.jxrObj.Nstep);
        ImgQualobj.jxrPSNR = zeros(1,ImgQualobj.jxrObj.Nstep);
            for i=1:ImgQualobj.jxrObj.Nstep;
                quality = quality + steps;
                quality_str = int2str(quality);
```

```
        jxrFile = [ImgQualobj.jxrObj.outputFile quality_str bmpTitle];
        I = rgb2gray(imread(jxrFile));
        ImgQualobj.jxrPSNR (1,i) = psnr(double(I),double(Iref));
    end % for i ..

quality = 0;
steps = ((ImgQualobj.jp2Obj.max_quality -ImgQualobj.jp2Obj.min_quality) /
ImgQualobj.jp2Obj.Nstep);
ImgQualobj.jp2PSNR = zeros(1,ImgQualobj.jp2Obj.Nstep);
    for i=1:ImgQualobj.jp2Obj.Nstep;
        quality = quality + steps;
        quality_str = num2str(quality);
        jp2File = [ImgQualobj.jp2Obj.outputFile quality_str bmpTitle];
        I = rgb2gray(imread(jp2File));
        ImgQualobj.jp2PSNR (1,i) = psnr(double(I),double(Iref));
    end % for i ..
end

%Calculate SSIM values of JPEG, JXR and JPEG2000 encoded images in workspace
function CalculateSSIMs (ImgQualobj)
bmpTitle = '.bmp';
Iref = imread([ImgQualobj.jpegObj.inputFile bmpTitle]);
Irefgray = rgb2gray(Iref);
imgSize = size(Iref);
ImgQualobj.width = imgSize(1);
ImgQualobj.height = imgSize(2);
ImgQualobj.dimension = imgSize(3);
quality = 0;
steps = ((ImgQualobj.jpegObj.max_quality -ImgQualobj.jpegObj.min_quality) /
ImgQualobj.jpegObj.Nstep);
ImgQualobj.jpegSSIM = zeros(1,ImgQualobj.jpegObj.Nstep);
    for i=1:ImgQualobj.jpegObj.Nstep;
        quality = quality + steps;
        quality_str = int2str(quality);
        jpegFile = [ImgQualobj.jpegObj.outputFile quality_str bmpTitle];
        I = (imread(jpegFile));
        Igray = rgb2gray(I);
        [ImgQualobj.jpegSSIM(1,i) ImgQualobj.jpegSSIMmap] = ssim_index(double(Igray),
        double(Irefgray));
    end % for i ..

quality = 0;
steps = ((ImgQualobj.jxrObj.max_quality -ImgQualobj.jxrObj.min_quality) /
ImgQualobj.jxrObj.Nstep);
ImgQualobj.jxrSSIM = zeros(1,ImgQualobj.jxrObj.Nstep);
    for i=1:ImgQualobj.jxrObj.Nstep;
        quality = quality + steps;
        quality_str = int2str(quality);
        jxrFile = [ImgQualobj.jxrObj.outputFile quality_str bmpTitle];
        I = (imread(jxrFile));
        Igray = rgb2gray(I);
        [ImgQualobj.jxrSSIM(1,i) ImgQualobj.jxrSSIMmap] = ssim_index(double(Igray),
        double(Irefgray));
    end % for i ..

quality = 0;
steps = ((ImgQualobj.jp2Obj.max_quality - ImgQualobj.jp2Obj.min_quality) /
ImgQualobj.jp2Obj.Nstep);
ImgQualobj.jp2SSIM = zeros(1,ImgQualobj.jp2Obj.Nstep);
    for i=1:ImgQualobj.jp2Obj.Nstep;
        quality = quality + steps;
        quality_str = num2str(quality);
        jp2File = [ImgQualobj.jp2Obj.outputFile quality_str bmpTitle];
        I = (imread(jp2File));
        Igray = rgb2gray(I);
        [ImgQualobj.jp2SSIM(1,i) ImgQualobj.jp2SSIMmap] = ssim_index(double(Igray),
        double(Irefgray));
    end % for i ..
end

%Compare psnr values of JPEG and JPEG XR encoded images
%Since Jpeg XR encoder has more quality levels during encoding
```

```
%operation, use Jpeg encoded psnr values as reference and iterate
%through Jpeg XR psnr values during the calculation.
function [JpegIDs JpegXrIDs] = FindClosestPSNRs(ImgQualobj)
ImgQualobj.targetPSNRs = zeros(1,ImgQualobj.jpegObj.Nstep);
ImgQualobj.targetJxrId = zeros(1,ImgQualobj.jpegObj.Nstep);
ImgQualobj.targetJpegId = zeros(1,ImgQualobj.jpegObj.Nstep);

jxrCnt = ImgQualobj.jxrObj.Nstep;
    for jpegCnt=1:ImgQualobj.jpegObj.Nstep;
        while(jxrCnt > 0)
            if (ImgQualobj.jxrPSNR(1,jxrCnt) > ImgQualobj.jpegPSNR(1,jpegCnt))
            ImgQualobj.targetPSNRs(1,jpegCnt) = ImgQualobj.jxrPSNR(1,jxrCnt);
            ImgQualobj.targetJxrId(1,jpegCnt) = jxrCnt;
            jxrCnt = jxrCnt - 1;
            break;
            end
            jxrCnt = jxrCnt - 1;
        end
        ImgQualobj.targetJpegId(1,jpegCnt) = jpegCnt;
    end
    JpegIDs = ImgQualobj.targetJpegId;
    JpegXrIDs = ImgQualobj.targetJxrId;
end

%Compare ssim values of JPEG and JPEG XR encoded images
%Since Jpeg XR encoder has more quality levels during encoding
%operation, use Jpeg encoded ssim values as reference and iterate
%through Jpeg XR ssim values during the calculation.
function [JpegIDs JpegXrIDs] = FindClosestSSIMs(ImgQualobj)
ImgQualobj.targetSSIMs = zeros(1,ImgQualobj.jpegObj.Nstep);
ImgQualobj.targetJxrId = zeros(1,ImgQualobj.jpegObj.Nstep);
ImgQualobj.targetJpegId = zeros(1,ImgQualobj.jpegObj.Nstep);

jxrCnt = ImgQualobj.jxrObj.Nstep;
    for jpegCnt=1:ImgQualobj.jpegObj.Nstep;
        while(jxrCnt > 0)
            if (ImgQualobj.jxrSSIM(1,jxrCnt) > ImgQualobj.jpegSSIM(1,jpegCnt))
            ImgQualobj.targetSSIMs(1,jpegCnt) = ImgQualobj.jxrSSIM(1,jxrCnt);
            ImgQualobj.targetJxrId(1,jpegCnt) = jxrCnt;
            jxrCnt = jxrCnt - 1;
            break;
            end
            jxrCnt = jxrCnt - 1;
        end
        ImgQualobj.targetJpegId(1,jpegCnt) = jpegCnt;
    end
    JpegIDs = ImgQualobj.targetJpegId;
    JpegXrIDs = ImgQualobj.targetJxrId;
end

%Draws PSNR results w.r.t bitsperpixel(bpp)
function plotPSNRandSSIMResults(ImgQualobj)
    jpegtitle = '.jpeg';
    jxrtitle = '.jxr';
    jp2title = '.jp2';
    ImgQualobj.jpegFileSizeVector = zeros(1, ImgQualobj.jpegObj.Nstep);
    ImgQualobj.jxrFileSizeVector = zeros(1, ImgQualobj.jxrObj.Nstep);
    ImgQualobj.jp2FileSizeVector = zeros(1, ImgQualobj.jp2Obj.Nstep);
    BPPjpeg = zeros(1, ImgQualobj.jpegObj.Nstep);
    BPPjxr = zeros(1, ImgQualobj.jxrObj.Nstep);
    BPPjp2 = zeros(1, ImgQualobj.jp2Obj.Nstep);

    scalerJpeg = (ImgQualobj.jpegObj.max_quality - ImgQualobj.jpegObj.min_quality)/
    ImgQualobj.jpegObj.Nstep;
    for i = 1:ImgQualobj.jpegObj.Nstep,
    i_str = int2str(i*scalerJpeg);
    filename = [ImgQualobj.jpegObj.outputFile i_str jpegtitle];
    s = dir(filename);
    ImgQualobj.jpegFileSizeVector(i) = int64(s.bytes);
    BPPjpeg(i) = (ImgQualobj.jpegFileSizeVector(i)*8)/(ImgQualobj.width*ImgQualobj.height);
    end
```

```
            scalerJxr = (ImgQualobj.jxrObj.max_quality - ImgQualobj.jxrObj.min_quality)/
            ImgQualobj.jxrObj.Nstep;
            for i = 1:ImgQualobj.jxrObj.Nstep,
            i_str = int2str(i*scalerJxr);
            filename = [ImgQualobj.jxrObj.outputFile i_str jxrtitle];
            s = dir(filename);
            ImgQualobj.jxrFileSizeVector(i) = int64(s.bytes);
            BPPjxr(i) = (ImgQualobj.jxrFileSizeVector(i)*8)/(ImgQualobj.width*ImgQualobj.height);
            end

            scalerJp2 = (ImgQualobj.jp2Obj.max_quality - ImgQualobj.jp2Obj.min_quality)/
            ImgQualobj.jp2Obj.Nstep;
            for i = 1:ImgQualobj.jp2Obj.Nstep,
            i_str = num2str(i*scalerJp2);
            filename = [ImgQualobj.jp2Obj.outputFile i_str jp2title];
            s = dir(filename);
            ImgQualobj.jp2FileSizeVector(i) = int64(s.bytes);
            BPPjp2(i) = (ImgQualobj.jp2FileSizeVector(i)*8)/(ImgQualobj.width*ImgQualobj.height);
            end

            %Since quality scale of JXR is in different direction w.r.t
            %JPEG, the PSNR, SSIM and BPP vectors shall be reverted.
            BPPjxr = BPPjxr(end:-1:1);
            jxrPSNRreverted = ImgQualobj.jxrPSNR(end:-1:1);
            jxrSSIMreverted = ImgQualobj.jxrSSIM(end:-1:1);

            figure;
            subplot(1,2,1);
            plot(BPPjpeg, ImgQualobj.jpegPSNR,'blue', BPPjxr, jxrPSNRreverted,'red', BPPjp2,
            ImgQualobj.jp2PSNR, 'green','LineWidth',2);
            title('PSNR vs BitsPerPixel'); % title
            ylabel('PSNR (dB)'); % label for y axis
            xlabel('BitsPerPixel (bpp)'); % label for x axis
            legend('JPEG','JPEG XR', 'JPEG 2000');
            legend('Location','SouthEast'); % move legend to lower right

            subplot(1,2,2);
            plot(BPPjpeg, ImgQualobj.jpegSSIM, 'blue', BPPjxr, jxrSSIMreverted, 'red', BPPjp2,
            ImgQualobj.jp2SSIM, 'green','LineWidth',2);
            title('SSIM vs BitsPerPixel'); % title
            ylabel('SSIM'); % label for y axis
            xlabel('BitsPerPixel (bpp)'); % label for x axis
            legend('JPEG','JPEG XR', 'JPEG 2000');
            legend('Location','SouthEast'); % move legend to lower right

        end


    end

    methods(Static)
        %Calculate PSNR of two images
        function PSNRval = CalculatePSNR(inputFile, refFile)
            I = rgb2gray(imread(inputFile));
            Iref = rgb2gray(imread(refFile));
            PSNRval = psnr(double(I), double(Iref));
        end

        %Calculate SSIM and SSIMmap of two images
        function [SSIMval SSIMmap] = CalculateSSIM(inputFile, refFile)
            I = rgb2gray(imread(inputFile));
            Iref = rgb2gray(imread(refFile));
            [SSIMval SSIMmap] = ssim_index(I, Iref);
        end



    end

end
```

ÖZGEÇMİŞ

Ad Soyad: Sezer BAĞLAN

Doğum Yeri ve Tarihi: İstanbul / 26.06.1986

Adres: Seyitnizam mah. Şehit Er Erkan Alyanak sok No:5/8 Zeytinburnu/İST.

E-Posta: sezerbaglan@gmail.com

Lisans: Orta Doğu Teknik Üniversitesi, Elektrik-Elektronik Mühendisliği (2008)

Mesleki Deneyim ve Ödüller:
- AR-GE Uzmanı, Arçelik Elektronik İşletmesi A.Ş. 2009 - ?
- AR-GE Mühendisi, Grundig Elektronik A.Ş. 2008 - 2009
- Koç Grubu Ödülleri, "PVR Özellikli LCD TV" projesi ödülü
- Koç Grubu Şirketleri 15. İnovasyon Günü patent buluşçusu ödülü

Yayın ve Patent Listesi:

TEZDEN TÜRETİLEN YAYINLAR/SUNUMLAR