Robert Bielas, Michał Ćwiertnia
Złożone Systemy Cyfrowe - projekt.
Dokumentacja.

Temat projektu:
Dekompresja plików graficznych w formacie jpg, jpg-ls, jpg 2000.

Cel:
Projekt jest zasadniczo wstępem/przygotowaniem do zaprogramowania zestawu uruchomieniowego DE-2 w ten sposób, aby mając na wejściu plik w formacie jpg, jpg-ls lub jpg 2000 dokonać jego dekompresji, a wynik wyświetlić na wyświetlaczu VGA. Głównym celem zadania w tym semestrze jest zrozumienie mechanizmów zachodzących w wymienionych standardach oraz napisanie kodu w C/C++ demonstrującego przykładowe implementacje dekoderów odpowiednich formatów.

Część pierwsza - standardowy JPG

Struktura jpg.
Format jpg jest zdefiniowany poprzez zbiór headerów(popularnie zwanymi markerami), z których każdy ma określoną składnię, znaczenie, pola i wartości określonych pól.
Markery mają sturkturę dwubajtową, z których pierwszy - ff - jest znacznikiem oznaczającym początek markeru, drugi zaś rodzaj markera.
Każdy plik rozpoczyna się poprzez marker początku pliku kóry ma wartość 0xffd8. Plik kończy się markerem 0xffd9.
Po znaczniku początku pliku występuje obszar metadanych. Jest to lista markerów o znaczeniu na poły informacyjnym, na poły opisowym. Najważniejsze dla nas są markery opisowe, które wymieniają najważniejsze charakterystyki obrazka, w tym jego długość, szerokość w pikselach, ilość komponentów graficznych, próbkowanie itp.
Tabelka poniżej pokazuje znaczenie poszczególnych znaczników.
Najważniejsze z punktu widzenia naszego projektu to xffc0, xffc4, xffda, xffdb.

xffc0 - marker który opisuje tryb wybrany w naszym projekcie.
Jego składnia: 2 bajty wskazujące długość tego markera (bez symbolu xffc0, razem z 2 bajtami opisującymi długość), bajt precyzji, 2 bajty ilości rzędów, 2 bajty ilości pikseli w rzędzie, bajt ilości komponentów (N) , a potem N 3 bajtowych opisów komponentów (id, samplowanie, klasa Huffmana).

xffdb - opis tabel(ek/ki) kwantyzacji wykorzystywanych przy dekompresji.
Składnia:
2 bajty opisujących długość M, ilość tabelek wyliczamy N = (M - 2) /(64 + 1)
N tabelek:
1 bajt, z którego pierwsze 4 bity to precyzja (0 - precyzja 8bitowa, 1- precyzja 12 bitowa, to zazwyczaj jest jednak 0), następne 4 bity opisujące id tabelki.
Następnie występują 64 bajty właściwej tabelki zapisanej zygzakiem.

xffc4 - znacznik tabelki Huffmana
2 bajtowa długość M,
1 bajt - pierwsze 4 bity komponentu używającego tabelki, następne 4 to klasa drzewa(0 - opisujące współczynniki DC, 1 - opisujące współczynniki AC).

Następnie N tabelek, gdzie

$$N = \frac{M-2}{\sum_{i=1}^{16} len(i) + 1}$$

len to ilość kodów i bitowych

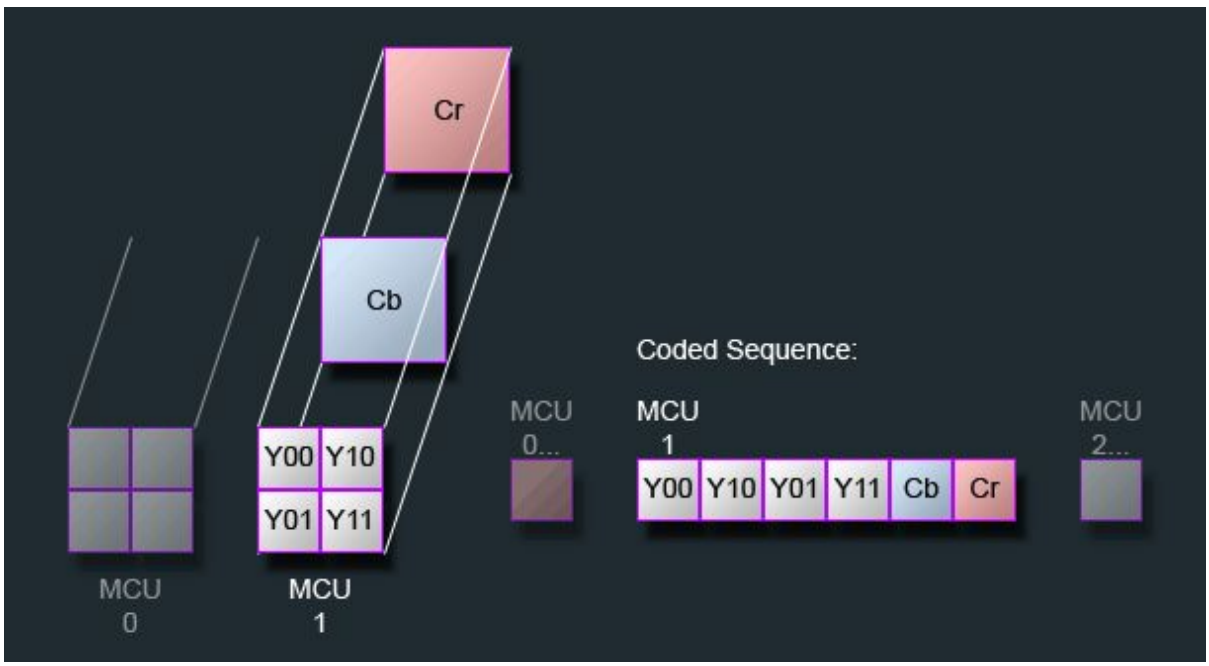
xffda - start of scan
składnia: 2 bajtowe długość N
Następnie 1 bajtowa ilość komponentów M i 3 bajtowe opisy tych komponentów w postaci:
pierwszy bajt - id komponentu,

drugi bajt - pierwsze 4 bity to id tabelki huffmana dla klasy 0(dc), następne 4 bity to id tabelki huffmana dla klasy 1(ac)
Następnie 3 bajty które są stałe dla naszego rodzaju kompresji.
Całość treści po tych znacznikach to już właściwy opis kolorów w pliku, zakończone ewentualnym dopełnieniem wymiarów pliku do rozmiarów będących wielokrotnością 8 i obowiązkowym znacznikiem 0xffd9 końca pliku.

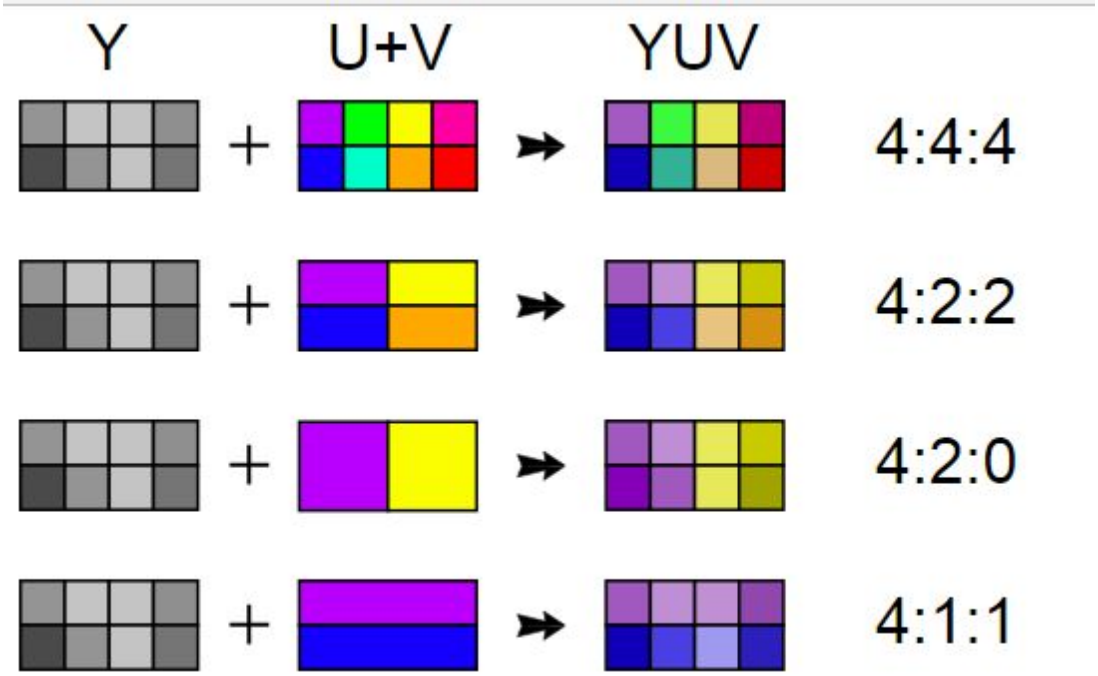
Code Assignment	Symbol	Description
Start Of Frame markers, non-differential, Huffman coding		
X'FFC0' X'FFC1' X'FFC2' X'FFC3'	SOF_0 SOF_1 SOF_2 SOF_3	Baseline DCT Extended sequential DCT Progressive DCT Lossless (sequential)
Start Of Frame markers, differential, Huffman coding		
X'FFC5' X'FFC6' X'FFC7'	SOF_5 SOF_6 SOF_7	Differential sequential DCT Differential progressive DCT Differential lossless (sequential)
Start Of Frame markers, non-differential, arithmetic coding		
X'FFC8' X'FFC9' X'FFCA' X'FFCB'	JPG SOF_9 SOF_{10} SOF_{11}	Reserved for JPEG extensions Extended sequential DCT Progressive DCT Lossless (sequential)
Start Of Frame markers, differential, arithmetic coding		
X'FFCD' X'FFCE' X'FFCF'	SOF_{13} SOF_{14} SOF_{15}	Differential sequential DCT Differential progressive DCT Differential lossless (sequential)
Huffman table specification		
X'FFC4'	DHT	Define Huffman table(s)
Arithmetic coding conditioning specification		
X'FFCC'	DAC	Define arithmetic coding conditioning(s)
Restart interval termination		
X'FFD0' through X'FFD7'	RST_m*	Restart with modulo 8 count "m"
Other markers		
X'FFD8' X'FFD9' X'FFDA' X'FFDB' X'FFDC' X'FFDD' X'FFDE' X'FFDF' X'FFE0' through X'FFEF' X'FFF0' through X'FFFD' X'FFFE'	SOI* EOI* SOS DQT DNL DRI DHP EXP APP_n JPG_n COM	Start of image End of image Start of scan Define quantization table(s) Define number of lines Define restart interval Define hierarchical progression Expand reference component(s) Reserved for application segments Reserved for JPEG extensions Comment
Reserved markers		
X'FF01' X'FF02' through X'FFBF'	TEM* RES	For temporary private use in arithmetic coding Reserved

Opis procesu. Etapy dekompresji.
Dekompresja jest procesem symetrycznie odwrotnym do procesu kompresji jpg. Aby odtworzyć wartości skompresowane w pliku formatu jpg, przede wszystkim należy rozszyfrować wartości zakodowane w strumieniu następującym po znaczniku xffda.
Obrazek jest koncepcyjnie podzielony na bloki 8x8 pikseli. Jest to najbardziej podstawowa jednostka podziału, z którego budowane są bardziej złożone struktury.
Omawianie tych struktur należy zacząć od omówienia procesu próbkowania.
Głównym celem jpga jest zmniejszenie rozmiarów obrazka korzystającego z przestrzeni kolorów RGB i wszystkie zabiegi, algorytmy oraz mechanizmy w tym formacie są tworzone właśnie z myślą o tym. Inżynierowie zajmujący się tym zagadnieniem wykorzystali zjawisko charakterystyczne dla ludzkiego oka: największy wkład w postrzeganie przez nas świata mają czopki, które są znacznie bardziej czułe na zmianę intensywności światła aniżeli sam kolor. Za odbiór kolorów w naszym oku odpowiedzialne są zdecydowanie mniej wrażliwe pręciki.
Pomysł polega na tym, aby przestać używać 3 bajtowej przestrzeni kolorów RGB i wykorzystać bardziej oszczędną, lecz równoważną przestrzeń YCbCr. odpowiadającą temu zjawisku. Y jest elementem(komponentem) odpowiadającym za reprezentację natężenia światła, tzw. luminacja. Cb i Cr to komponenty chrominancji (domieszki koloru) odpowiednio niebieskiego i czerwonego koloru. Domieszka zieleni nie jest potrzebna, gdyż kolor zielony w przestrzeni RGB jest obliczalny przy użyciu jedynie tych trzech powyższych wartości.

Wykorzystując tą przestrzeń, wprowadza się proces próbkowania. Ogólnie mówiąc na kilka komponentów Y przypada jeden lub dwa elementy chrominancji.
W naszym parserze obsługujemy najbardziej powszechne próbkowanie 4:2:0, co oznacza, że na 4 wartości komponentu Y(ułożone w kwadrat 4 bloki 8x8 pikseli) przypada 1 komponent Cr i 1 komponent Cb.



Taki układ możemy zgrupować w nieco bardziej złożoną strukturę zwaną MCU (minimal coded unit).



Obrazek: Wkład poszczególnych komponentów w przestrzeni YCbCr w różnych trybach próbkowania.

Każdy z komponentów MCU to blok 8x8 pikseli. Taki układ nie jest niczym przypadkowym, gdyż dzięki temu możemy te komponenty poddać na wejście powszechnie używanego w kompresji formatu jpg przekształcenia macierzowego jakim jest dyskretna transformata kosinusowa, którą definiujemy w następujący sposób:

$$U = \frac{1}{2}\begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \cos\frac{\pi}{16} & \cos\frac{3\pi}{16} & \cos\frac{5\pi}{16} & \cos\frac{7\pi}{16} & \cos\frac{9\pi}{16} & \cos\frac{11\pi}{16} & \cos\frac{13\pi}{16} & \cos\frac{15\pi}{16} \\ \cos\frac{2\pi}{16} & \cos\frac{6\pi}{16} & \cos\frac{10\pi}{16} & \cos\frac{14\pi}{16} & \cos\frac{18\pi}{16} & \cos\frac{22\pi}{16} & \cos\frac{26\pi}{16} & \cos\frac{30\pi}{16} \\ \cos\frac{3\pi}{16} & \cos\frac{9\pi}{16} & \cos\frac{15\pi}{16} & \cos\frac{21\pi}{16} & \cos\frac{27\pi}{16} & \cos\frac{33\pi}{16} & \cos\frac{39\pi}{16} & \cos\frac{45\pi}{16} \\ \cos\frac{4\pi}{16} & \cos\frac{12\pi}{16} & \cos\frac{20\pi}{16} & \cos\frac{28\pi}{16} & \cos\frac{36\pi}{16} & \cos\frac{44\pi}{16} & \cos\frac{52\pi}{16} & \cos\frac{60\pi}{16} \\ \cos\frac{5\pi}{16} & \cos\frac{15\pi}{16} & \cos\frac{25\pi}{16} & \cos\frac{35\pi}{16} & \cos\frac{45\pi}{16} & \cos\frac{55\pi}{16} & \cos\frac{65\pi}{16} & \cos\frac{75\pi}{16} \\ \cos\frac{6\pi}{16} & \cos\frac{18\pi}{16} & \cos\frac{30\pi}{16} & \cos\frac{42\pi}{16} & \cos\frac{54\pi}{16} & \cos\frac{66\pi}{16} & \cos\frac{78\pi}{16} & \cos\frac{90\pi}{16} \\ \cos\frac{7\pi}{16} & \cos\frac{21\pi}{16} & \cos\frac{35\pi}{16} & \cos\frac{49\pi}{16} & \cos\frac{63\pi}{16} & \cos\frac{77\pi}{16} & \cos\frac{91\pi}{16} & \cos\frac{105\pi}{16} \end{bmatrix}$$

Elements of the 8 x 8 DCT matrix.

Zastosowanie tego narzędzia to kolejny przykład ukierunkowania twórców formatu na maksymalną kompresję pliku graficznego. Ma ono bardzo przydatną właściwość: piksele w bloku zostają przetransformowane z domeny przestrzennej do domeny częstotliwościowej o takiej własności, że najbardziej dominująca częstotliwość jest sprowadzana do lewego górnego rogu macierzy (Czyli elementu nr 1 macierzy), zaś reszta to generalnie wartości równe zero lub bliskie zeru.

Pierwszy element nazywamy współczynnikiem DC dyskretnej transformaty kosinusowej(direct current), pozostałe 63 to współczynniki AC.
Poniżej wizualizacja wyniku DCT:

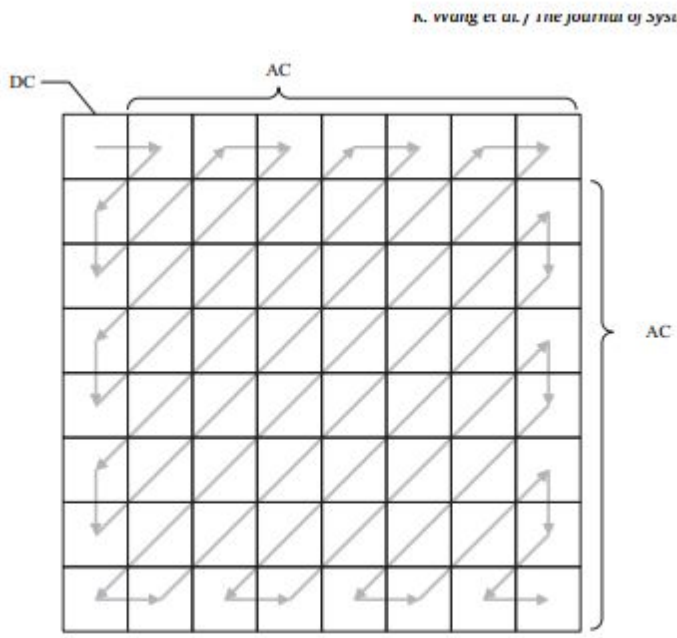


Fig. 3. Zigzag sequence.

Strzałkami oznaczono w jaki sposób części komponentów są kodowane od postaci dwuwymiarowej macierzy do jednowymiarowego strumienia danych - tzw. zigzag order.
Mając na uwadze informacje dot. układu komponentów w trybie próbkowania 4:2:0 w pojedynczym MCU, należy przejść do etapu kodowania, jakim jest tzw. entropy coding. Najbardziej popularnym (niekomercyjnym i intuicyjnym) tego typu kodowaniem jest kodowanie Huffmana. W baseline DCT przy użyciu tego szyfru kodujemy komponenty następująco: pierwszy element (dc) bloku komponentu jest kodowany przy użyciu drzewa huffmana klasy 0.
Kolejne 63 elementy są opisane przez sekwnecję hc1 hv1 ... hcn hvn 0x00.
hci to skrót od huffman code o indeksie i, hvi oznacza huffman value o indeksie i, dotyczące drzewa huffmana klasy 1(ac). Koniec bloku współczynników ac jest oznaczane bajtem EOB(end of block) o wartości 0x00. Jeżeli jesteśmy na indeksie mniejszym niż 63 gdy występuje EOB, to pozostałe elementy bloku wypełniamy zerami.

Wartości hvi to jednobajtowe kody, w których: pierwsze 4 bity oznaczają ilość kolejnych elementów komponentu będącymi zerami(RZL - run zero length), zaś następne 4 bity wartość długości w bitach n już właściwej wartości v, która następuje w macierzy komponentu po tej sekwencji zer, a która jest zakodowana dalej w strumieniu. Dla przypomnienia wszystkie elementy bloku są zapisywane w porządku zig zag.
Ta wartość v jest kolejnym rodzajem kodu: bezpośrednią wartość elementu komponentu v' możemy odczytać z poniższej tabelki w zależności od wartości n i v i ją właśnie wpisać po sekwencji RZL.
Uwaga: element dc odszyfrowujemy korzystając z tej samej tabelki, lecz siłą rzeczy pierwsze 4 bity hv będą tam zawsze wyzerowane.

DC Code	Size	Additional Bits		DC Value	
00	0			0	
01	1	0	1	-1	1
02	2	00,01	10,11	-3,-2	2,3
03	3	000,001,010,011	100,101,110,111	-7,-6,-5,-4	4,5,6,7
04	4	0000,...,0111	1000,...,1111	-15,...,-8	8,...,15
05	5	0 0000,...	...,1 1111	-31,...,-16	16,...,31
06	6	00 0000,...	...,11 1111	-63,...,-32	32,...,63
07	7	000 0000,...	...,111 1111	-127,...,-64	64,...,127
08	8	0000 0000,...	...,1111 1111	-255,...,-128	128,...,255
09	9	0 0000 0000,...	...,1 1111 1111	-511,...,-256	256,...,511
0A	10	00 0000 0000,...	...,11 1111 1111	-1023,...,-512	512,...,1023
0B	11	000 0000 0000,...	...,111 1111 1111	-2047,...,-1024	1024,...,2047

Bezpośrednio po odczytaniu jawnych wartości współczynników dc i ac z powyższej tabelki należy użyć tzw. tabeli kwantyzacji. Zazwyczaj są dwie: jedna odpowiada elementom komponentu luminancji, zaś druga elementom Cr i Cb. Przy kompresji współczynniki dzieli się przez odpowiednie elementy w tabelce kwantyzacji (dobranej tak, by parametry obrazka były optymalne), zaś przy dekompresji - wymnaża się je przez siebie.

Zdekodowane współczynniki DC w komponentach nie są jednak jeszcze ostateczne, tylko zapisane jako różnica w stosunku do elementu dc poprzedniego komponentu odpowiedniego rodzaju w porządku w jakim pojawiły się w zakodowanym strumieniu. Tylko pierwsze komponenty każdego rodzaju mają w strumieniu zakodowaną bezwzględną wartość elemtnu dc (czyli od nich nic nie odejmujemy).

Wzory do przejścia z przestrzeni YCbCr do przestrzeni RGB są następujące:
Red = Y + 1.402 * Cr + 128
Green = Y - 0.3437 * Cb - 0.7143 * Cr + 128
Blue = Y + 1.772 * Cb + 128

Biorąc to wszystko pod uwagę, dekompresja (po sparsowaniu metadaty po odpowiednich markerach, odczytaniu specyfikacji tabelek huffmana i zbudowaniu w oparciu o nie drzew, oraz zainstalowaniu wszystkich tabel kwantyzacji) polega więc kolejno na:
1. Zidentyfikowaniu rodzaju próbkowania - domyślnie 4:2:0;
2. dla każdego koponentu:
    - odczytać współczynnik dc przy użycia drzewa klasy 0 i powyższej tabelki, wpisać go do komponentu
    - odczytać sekwencję hc1 hv1 .. hcn hvn 0x00 i w opisany wyżej sposób zdekodować i wpisać do komponentu, używając porządku zig-zag.
    - dla każdego komponentu C użyć tabelki kwantyzacji Q o odpowiednim id(domyślnie - id tabelki kwantyzacji komponentu luminancji wynosi 0, dla komponentu chrominancji 1) i wymnożyć je przez siebie tak, że powstaje nowa macierz C' której elementy C'(i)=C(i)*Q(i).
3. Odczytać bezwzględne wartości DC wszystkich komponentów we wszystkich MCU.
4. Dokonać Inverse DCT, czyli przejścia z domeny częstotliwościowej do domeny przestrzennej.
5. Przejść z przestrzeni YCbCr do RGB dla każdego komponentu we wszystkich MCU.
6. Utworzyć trzy tabele R',G' i B' o rozmiarach odczytanych po znaczniku 0xffc0
7. Ułożyć elementy RGB w każdym MCU w odpowiadającym im miejscach w tablicach R'G' i B'
8. dostosować wartości w tabelach R' G' i B' tak by były zawarte w przedziale (0,255)

Bibliografia:
itu-t 81 INFORMATION TECHNOLOGY – DIGITAL COMPRESSION AND CODING OF CONTINUOUS-TONE STILL IMAGES – REQUIREMENTS AND GUIDELINES
http://www.impulseadventure.com/photo/jpeg-huffman-coding.html
http://www.impulseadventure.com/photo/jpeg-decoder.html
http://www.whydomath.org/node/wavlets/basicjpg.html

Część druga - JPEG 2000

Opis procesu dekodowania JPEG-a 2000.

1. Struktura bitowa JPG-2000
Format jpg jest zdefiniowany poprzez zbiór headerów(popularnie zwanymi markerami), z których każdy ma określoną składnię, znaczenie, pola i wartości określonych pól. Markery mają sturkturę dwubajtową, z których pierwszy - ff - jest znacznikiem oznaczającym początek markeru, drugi zaś rodzaj markera. Każdy plik rozpoczyna się poprzez marker początku pliku kóry ma wartość 0xffd8. Plik kończy się markerem 0xffd9. Po znaczniku początku pliku występuje obszar metadanych. Jest to lista markerów o znaczeniu na poły informacyjnym, na poły opisowym. Najważniejsze dla nas są markery opisowe, które wymieniają najważniejsze charakterystyki obrazka, w tym jego długość, szerokość w pikselach, ilość komponentów graficznych itp. Tabelka poniżej pokazuje znaczenie poszczególnych znaczników.

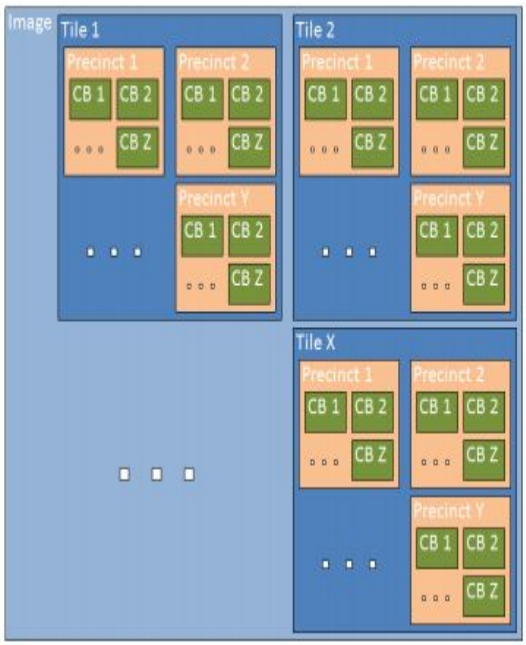
	Symbol	Code	Main header	Tile-part header
Delimiting markers and marker segments				
Start of codestream	SOC	0xFF4F	required[a]	not allowed
Start of tile-part	SOT	0xFF90	not allowed	required
Start of data	SOD	0xFF93	not allowed	last marker
End of codestream	EOC	0xFFD9	not allowed	not allowed
Fixed information marker segments				
Image and tile size	SIZ	0xFF51	required	not allowed
Functional marker segments				
Coding style default	COD	0xFF52	required	optional
Coding style component	COC	0xFF53	optional	optional
Region-of-interest	RGN	0xFF5E	optional	optional
Quantization default	QCD	0xFF5C	required	optional
Quantization component	QCC	0xFF5D	optional	optional
Progression order change[b)]	POC	0xFF5F	optional	optional
Pointer marker segments	TLM	0xFF55	optional	not allowed
Tile-part lengths	PLM	0xFF57	optional	not allowed
Packet length, main header	PLT	0xFF58	not allowed	optional
Packet length, tile-part header	PPM	0xFF60	optional	not allowed
Packed packet headers, main header[c)]	PPT	0xFF61	not allowed	optional
Packed packet headers, tile-part header[c)]	TLM	0xFF55	optional	not allowed
In-bit-stream markers and marker segments				
Start of packet	SOP	0xFF91	not allowed	not allowed in tile-part header, optional in-bit stream
End of packet header	EPH	0xFF92	optional inside PPM marker segment	optional inside PPT marker segment or in-bit stream
Informational marker segments				
Component registration	CRG	0xFF63	optional	not allowed
Comment	COM	0xFF64	optional	optional

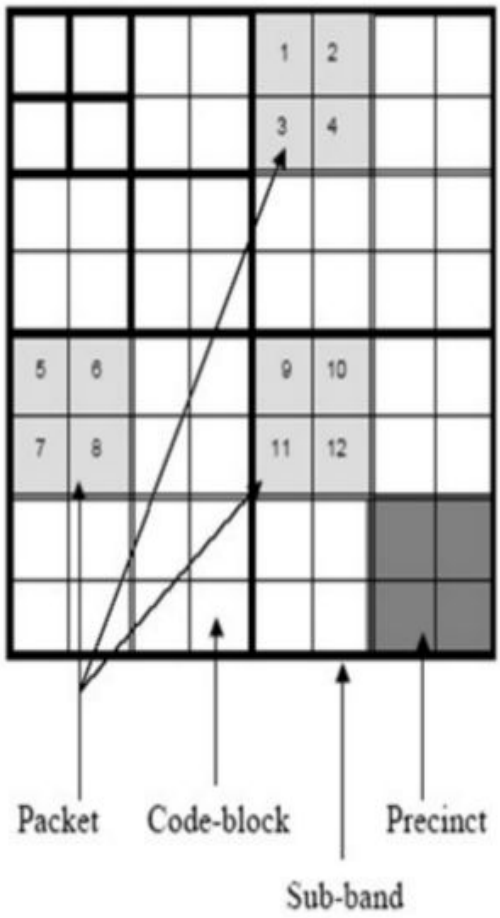
Jedynymi wymaganymi markerami nagłówkowymi są: SOC(Start of Codestream), SIZ(image and tile size), COD(coding style default) i QCD(Quantization default)., ponieważ zawierają one minimalną ilość informacji potrzebną, żeby całkowicie zdekodować obrazek. Pozostałe

albo mówią o zmianie wartości już ustawionych przez obowiązkowe headery lub noszą w sobie nieobowiązkową informację przykładowo dotyczącą przyspieszenia procesu dekodowania.

Marker SIZ zawiera wysokość i szerokość obrazka i "kafelków" (ang. tiles), ilość komponentów, współczynnik próbkowania i głębię bitową każdego komponentu.
Marker COD zawiera styl kodowania (stwierdza, czy pakiety mają maksymalny rozmiar itp.), progression order, ilość warstw poziomów dekompozycji, jaka odmiana/filtr wavelet transform jest stosowany oraz rozmiary tzw. paczek i bloków.
Marker SOT zawiera niezbędne informacje odnośnie kafelka, włącznie z jego indeksem, i rozmiar kafelka w bajtach. W pewnym miejscu za SOT musi wystąpić marker SOD (start of data), sygnalizujący koniec markera SOT i początek zakodowanego strumienia obrazka.
Jeśli parser nie odnajdzie markera SOC, wtedy zwracany jest błąd.

2. Hierarchia elementów pliku.
Struktura JPG2000 jest hierarchiczna: elementy są zgrupowane w komponenty(components), kafelki (tiles), obszary (precincts), na końcu zaś w bloki kodu (code blocks).

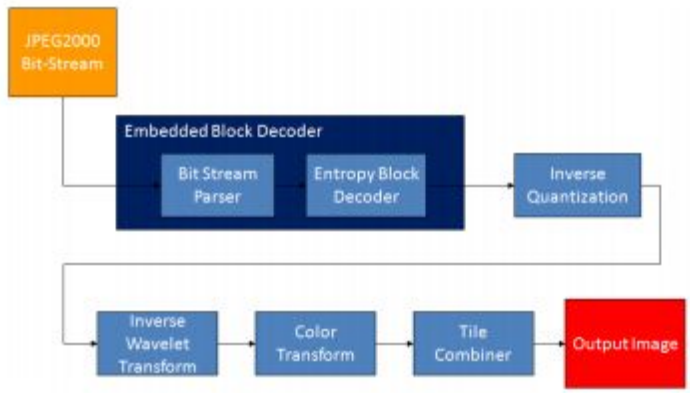




Powyższe rysunki obrazują sposób organizacji hierarchii elementów.
Obszar Image reprezentuje pojedynczy komponent, z jakich składa się obrazek. W przypadku gdy mamy do czynienia z obrazkiem monochromatycznym (w grayscale) komponent jest dokładnie jeden - w naszym projekcie skupiliśmy się jednakże na obrazku w przestrzeni kolorów RGB - zakładamy, że obszar Image występuje dokładnie trzykrotnie.
Najwyższym elementem w hierarchii jest tile. Gdy tego typu komponentów w obrazku występuje więcej niż jeden, w strumieniu są one odpowiednio oznaczone przez nagłówki SOT.
Każdy tile agreguje tzw. precincts (czyli zbiór bloków). Blok jest najniższym elementem hierarchii.
Informacje o zakodowanym obrazku są dostarczane do dekodera w postaci wyżej wspomnianego strumienia. Podstawową, spójną jednostką odczytywaną ze strumienia jest paczka (packet) - może ona zawierać ciąg bitów reprezentujących (w zależności od kontekstu) blok, obszar, lub nawet cały kafelek i komponent. Takie skrajne sytuacje zdarzają się, gdy rozmiary wejściowego obrazka są odpowiednio małe.

Paczki posiadają nagłówek i wnętrze. Nagłówek pakietu zawiera informacje o tym, czy zakodowany element w rzeczywistości zawiera jakiekolwiek dane (w skrajnych sytuacjach, gdy odpowiedni bit nagłówka paczki jest ustawiony na zero, pakiet nie wnosi niczego nowego do obrazka), głębię bitową każdego elementu, ilość tzw. decoding passess wykonywanych na każdym elemencie, i długość w bajtach każdego elementu. Po zebraniu wszystkich potrzebnych informacji z nagłówków, parser lokalizuje bitstream dla każdego elementu i przekazuje każdy z nich do Entropy Decodera.
Większość koderów JPEG-a 2000 przyjmuje następujące uproszczenia: ilość kafelków jest równa jeden, zaś obszar jest utożsamiany z blokiem. W projekcie zgodziliśmy się na te uproszczenia.

3. Proces dekodowania.
Rysunek poniżej przedstawia najważniejsze etapy dekodowania pliku graficznego zapisanego w formacie .jp2:

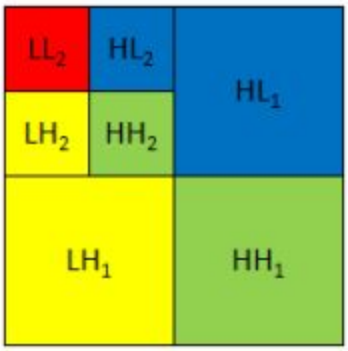


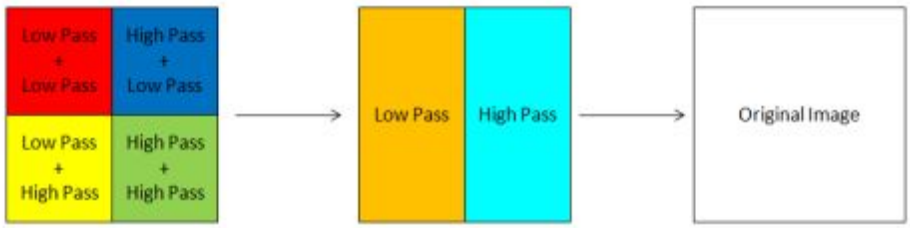
Poszczególne elementy w powyższym schemacie zostały opisane poniżej:

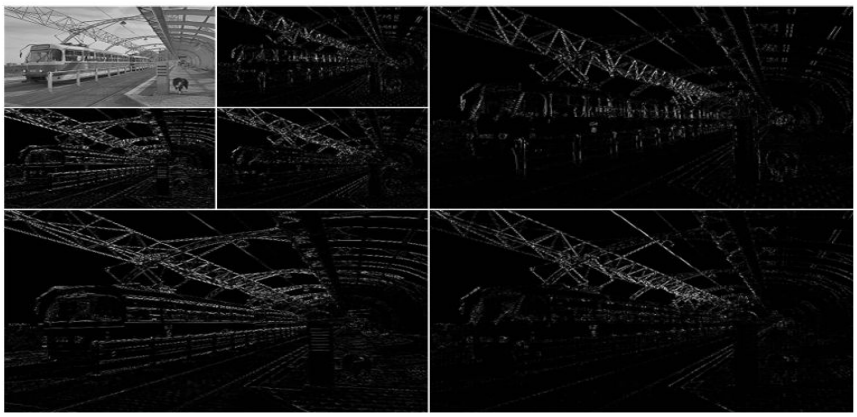
Bit Stream Parser:
Wydziela ze strumienia kolejne paczki. Każdą paczkę przetwarza, czytając najpierw jej nagłówek i znajdując w nim kluczowe informacje dotyczące elementu, następnie tworząc na ich podstawie strukturę/obiekt bloku. Taki obiekt jest wejściem dla Entropy Decodera. BSP współpracuje z automatem znanym w standardzie jako Tag Tree Decoder.

Entropy Block Decoder:
Na wejściu dostaje obiekt/strukturę reprezentującą blok, w której buforach przechowywane są skompresowane dane obrazka i informacje opisujące położenie tych danych w odniesieniu do początku (lewego górnego rogu) obrazka. ED współpracuje z arytmetycznym dekoderem (znany w standardzie jako MQ Decoder). Dekompresja na tym etapie składa się z faz (zwanych warstwami) i etapów(zwanych coding passes). Informacja o etapach jest unikalna dla każdego bloku, odczytuje się ją z nagłówka paczki. Ilość warstw natomiast uzyskuje się na etapie parsowania metadanych. UWAGA: kodery JPEG-a 2000 przyjmują uproszczenie polegające na tym, że ilość faz(warstw) wynosi jeden. W projekcie zgodziliśmy się na to uproszczenie.
Dekoder w odpowiedniej ilości etapów dekompresuje dane w buforze wejściowym, i zapisuje je w zaalokowanym buforze wyjściowym bloku.

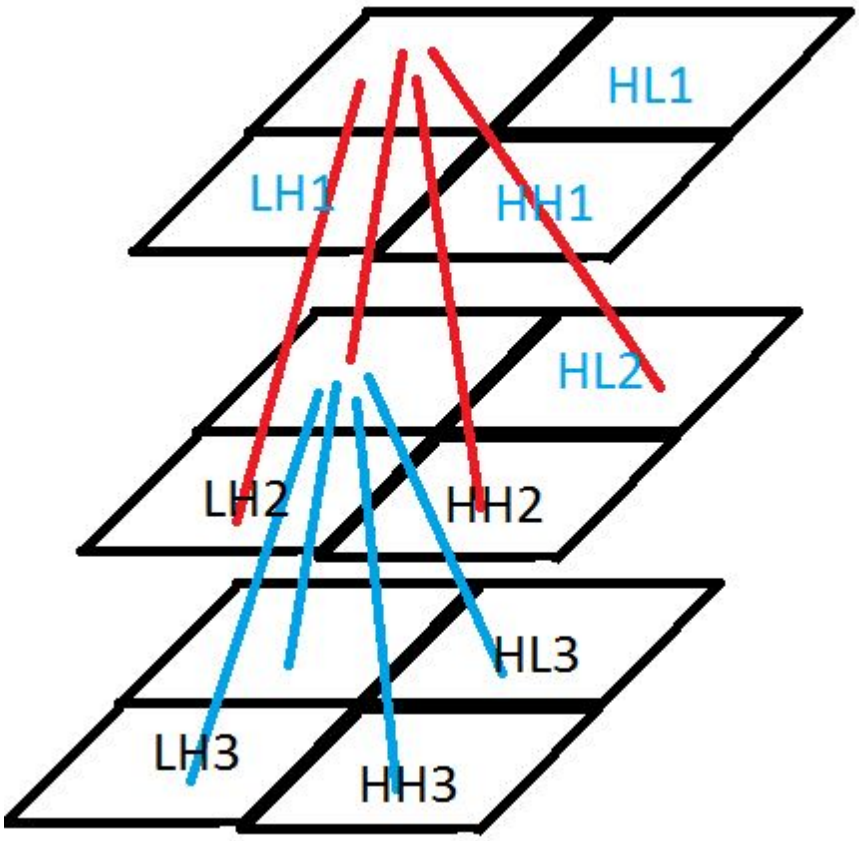
Inverse Wavelet Transform:
Dokonuje rekompozycji obrazka ze struktury drzewiastej do struktury płaskiej. Wprowadzamy definicję subband: jest to jeden z czterech obszarów na który Wavelet Transform podzielił pierwotny band. Band to poziom dekompozycji, który został dalej zdekomponowany. Wizualizuje to rysunek nr 1. Subbandy dostają nazwy (w kierunku rastrowym): LL, HL, LH, HH. L jest skrótem od Low Pass, natomiast H - od High Pass Wavelet Transform. Rysunek drugi pokazuje proces rekompozycji jednego bandu. Rysunek trzeci pokazuje, jak wygląda przykładowy obrazek po zastosowaniu dwóch przebiegów WT (pokazujemy to, aby dać pewną intuicję tego, z czym pracujemy).







Należy zwrócić uwagę na to, w jakim kierunku dokonuje się dekompozycji w WT na trzecim rysunku: zarys obrazka(najmniejsza rozdzielczość) jest skupiony w LL na każdym poziomie dekompozycji. Idąc w głąb drzewa podziału, w LL skupiamy coraz mniej szczegółów (pozostaje jedynie ogólny “zarys” obrazka). Dekomponuje się jedynie subband LL, uzyskując strukturę zobrazowaną na poniższym rysunku.



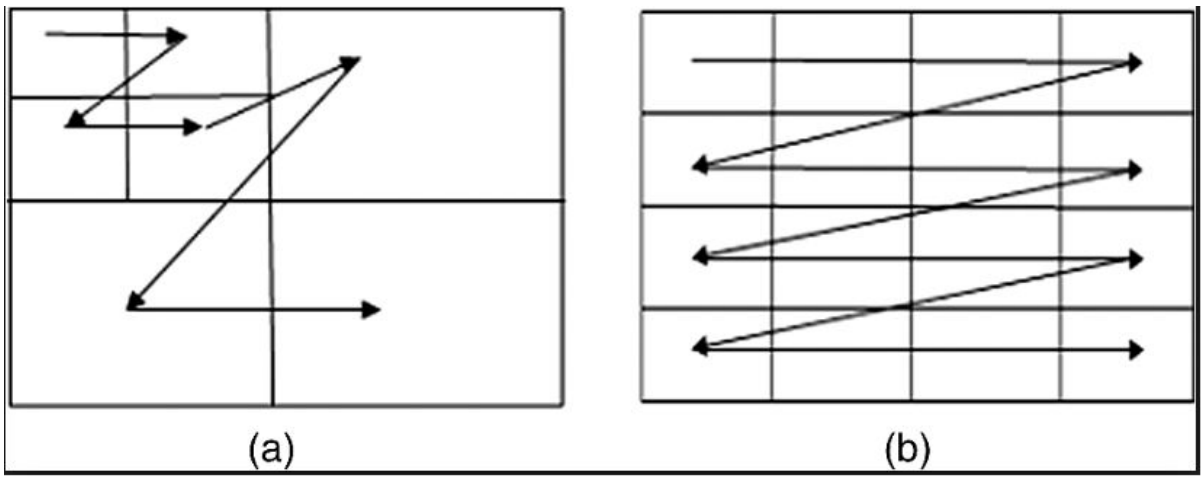
Szczegóły są podzielone/rozdystrybuowane na każdym poziomie pomiędzy HL, LH i HH.

Inverse Quantization:
Proces dekwantyzacji wygląda podobnie jak miało to miejsce w “podstawowym” JPGu - mając macierz bloku, do każdego jej elementu dodajemy wartość odjętą przez koder.

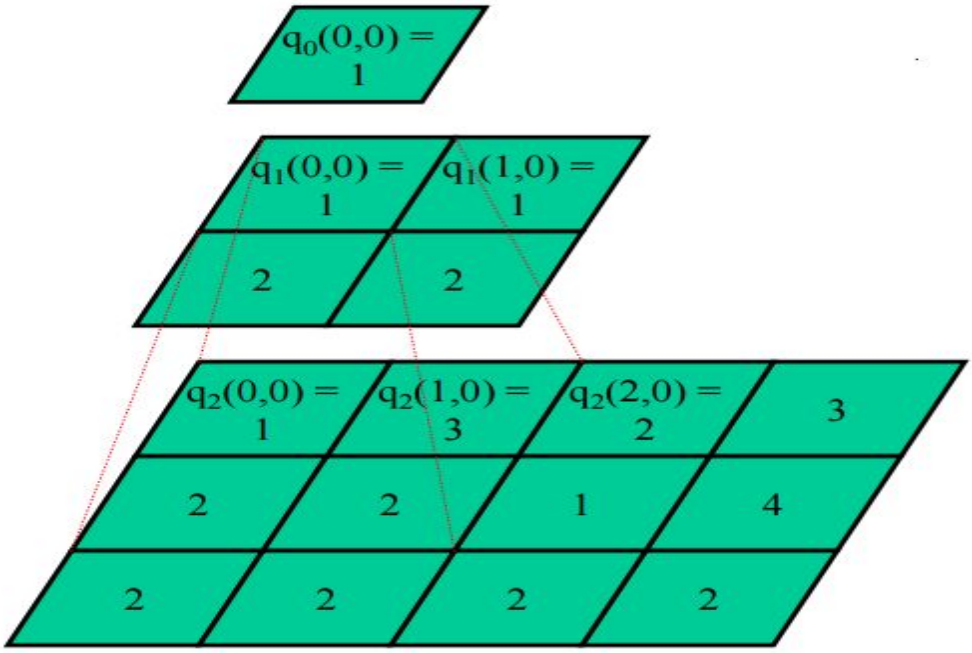
Color transform.
Przejście z jednej przestrzeni kolorów do drugiej. W naszym przypadku przechodzimy z YCbCr do RGB przy wykorzystaniu wzorów:
Red = Y + 1.402 * Cr + 128
Green = Y - 0.3437 * Cb - 0.7143 * Cr + 128
Blue = Y + 1.772 * Cb + 128

Tile Combiner
W przypadku gdy zdefiniowanych jest więcej niż jeden kafelek. ten moduł jest odpowiedzialny za ich ostateczny "montaż". Nie występuje on jednak w naszym projekcie, gdyż przyjęliśmy, że ilość kafelek = 1.

4. Szczegóły dekodowania. Przepływ kontroli.
Na samym początku następuje parsowanie metadanych przez MetaDataDecoder. Precyzuje on najważniejsze informacje dot. obrazka, jak jego wymiary, ilość poziomów dekompozycji, ilość komponentów(domyślnie 3) itd. jak zostało to opisane wcześniej w punkcie dot. markerów.
Wykorzystując informację o ilości poziomów dekompozycji i ilości komponentów tworzymy trzyelementową tablicę wskaźników na drzewa dekompozycji (Subband). Trzy elementy odpowiadają odpowiednio drzewom luminancji i chrominancji. Każdy obiekt Subband posiada listę bloków, które się na niego składają, a także wskazania na swoje dzieci, jeśli takowe posiada, oraz wskazanie na rodzica. Tak utworzona struktura danych jest przekazywana do dekoderów, którzy żądają od niej kolejnych bloków. Kolejność przekazywanych bloków jest ukazana poniżej (a - kolejność przechodzenia pom. subbandami, b - kolejność uzyskiwania bloków w subbandzie - kolejność rastrowa):

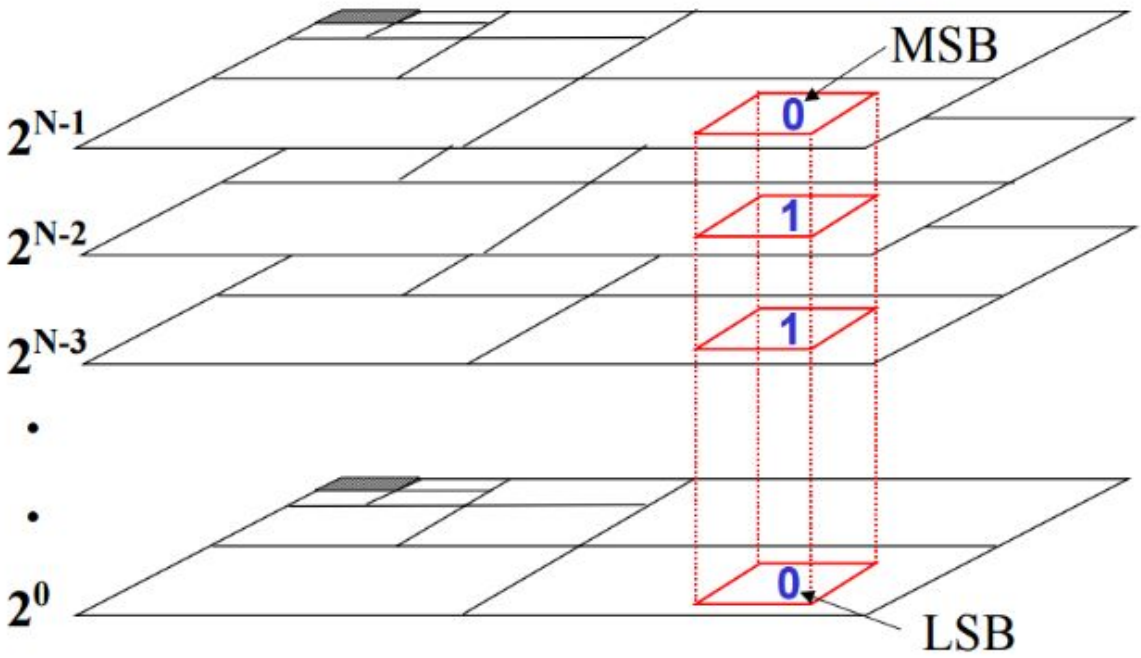


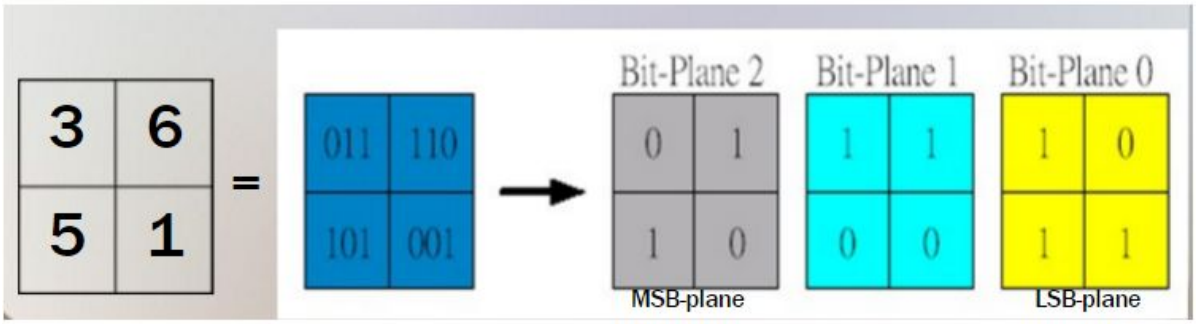
Przy założeniu, że rozmiary obrazka są potęgą dwójki, rozmiary kolejnych subbandów to rozmiary rodzica prawy shift o jeden bit. Korzystając z informacji uzyskanych z metadanych, poznajemy domyślne rozmiary bloku. Jeśli subband posiada rozmiary mniejsze niż domyślne rozmiary bloku, subband posiada jeden blok o rozmiarach równych swoich. W przeciwnym przypadku utrzymuje listę bloków o domyślnych rozmiarach. UWAGA: wzorem “podstawowego” JPEG-a, domyślne rozmiary bloków to 8x8 pikseli.
Iterując po blokach, na samym początku uruchamia się PacketHeaderDecoder, który wyłuskuje ze strumienia kolejne paczki, odczytuje informacje dotyczące bloku, i wypełnia bufor bloku na podstawie tych informacji. Informacje te zawierają zakodowaną liczbę ilości “zerowych” bit plane’ów, coding pass (etapów) i długości(w bajtach) bloku w strumieniu.
Aby zdekodować te informacje, dekoder korzysta z pomocniczego TagTreeDecoder. Algorytm używany w tym miejscu opiera się na następującej zasadzie: zaczynając od minimalnej wartości = 1, odczytujemy kolejne bity. bit równy zero oznacza, że wartość należy zinkrementować. Bit = 1 oznacza, że osiągnęliśmy wartość właściwą.



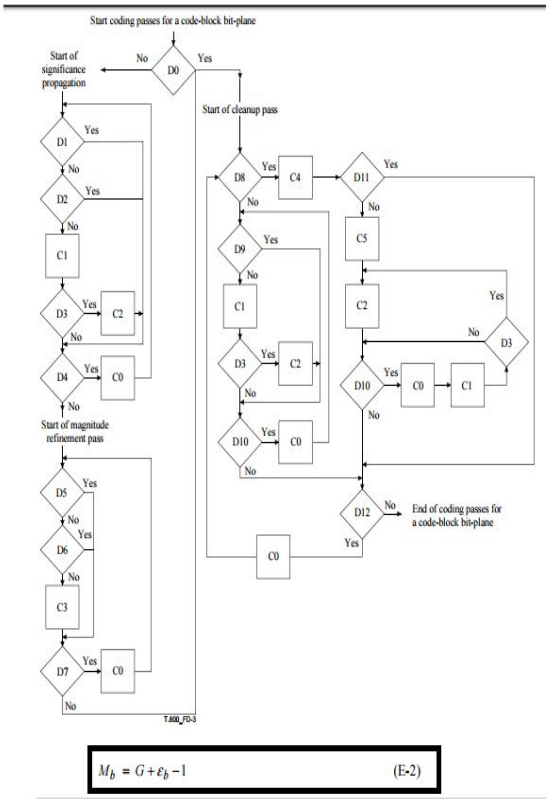
Rysunek przedstawia poglądowe działanie TagTreeDecodera.

Tak wypełniony blok jest przekazywany do EntropyDecodera.
Informacje zawarte w bloku są poddawane etapom zwanym coding pass.
Operujemy tutaj na pojęciu bitplane'ów. Bit plane jest to dwu-wymiarowa tablica bitowa, która zawiera bity współczynników WT TEJ SAMEJ WAŻNOŚCI. Obrazuje to rysunek poniżej.





Algorytm Entropy Dekodera pokazano na poniższym schemacie.



$$M_b = G + \varepsilon_b - 1 \qquad \text{(E-2)}$$

Table D.10 – Decisions in the context model flow chart

Decision	Question	Description
D0	Is this the first significant bit-plane for the code-block?	See D.3
D1	Is the current coefficient significant?	See D.3.1
D2	Is the context bin zero? (see Table D.1)	See D.3.1
D3	Did the current coefficient just become significant?	See D.3.1
D4	Are there more coefficients in the significance propagation?	
D5	Is the coefficient insignificant?	See D.3.3
D6	Was the coefficient coded in the last significance propagation?	See D.3.3
D7	Are there more coefficients in the magnitude refinement pass?	
D8	Are four contiguous undecoded coefficients in a column each with a 0 context?	See D.3.4
D9	Is the coefficient significant or has the bit already been coded during the Significance Propagation coding pass?	See D.3.4
D10	Are there more coefficients remaining of the four column coefficients?	
D11	Are the four contiguous bits all zero?	See D.3.4
D12	Are there more coefficients in the cleanup pass?	

Table D.11 – Decoding in the context model flow chart

Code	Decoded symbol	Context	Brief explanation	Description
C0	–	–	Go to the next coefficient or column	
C1	Newly significant?	Table D.1, 9 context labels	Decode significance bit of current coefficient (significance propagation or cleanup)	See D.3.1
C2	Sign bit	Table D.3, 5 context labels	Decode sign bit of current coefficient	See D.3.2
C3	Current magnitude bit	Table D.4, 3 context labels	Decode magnitude refinement pass bit of current coefficient	See D.3.3
C4	0 1	Run-length context label	Decode run-length of four zeros Decode run-length not of four zeros	See D.3.4
C5	00 01 10 11	UNIFORM	First coefficient is first with non-zero bit Second coefficient is first with non-zero bit Third coefficient is first with non-zero bit Fourth coefficient is first with non-zero bit	See D.3.4 and Table C.2

Istotną informacją jest to, w jaki sposób uzyskiwany jest kontekst przekazywany do MQ Decoder. Aby go uzyskać, korzystamy z zahardkodowanych LUT (patrz: konstruktor EntropyDecoder'a). Niech X będzie aktualnie rozpatrywanym współczynnikiem WT. Kontekst uzyskuje się, patrząc na ważność sąsiadów, których układ pokazano poniżej.

D_0	V_0	D_1
H_0	X	H_1
D_2	V_1	D_3

T.800_FD-2

Figure D.2 – Neighbors states used to form the context

Odczytując ilość bit-plane'ów ze wzoru (E-2) powyżej, a także znając odczytaną ilość coding pass, dekoder iteruje po odpowiednich SERIACH coding pass. Te serie wyglądają następująco: pierwsza iteracja składa się wyłącznie z cleanup pass. Pozostałe serie to sekwencje zero coding pass(inaczej significance pass), magnitude refinement pass oraz cleanup pass. Z każdą z tych coding pass jest związany także sign pass. Definicje znajdują się poniżej:

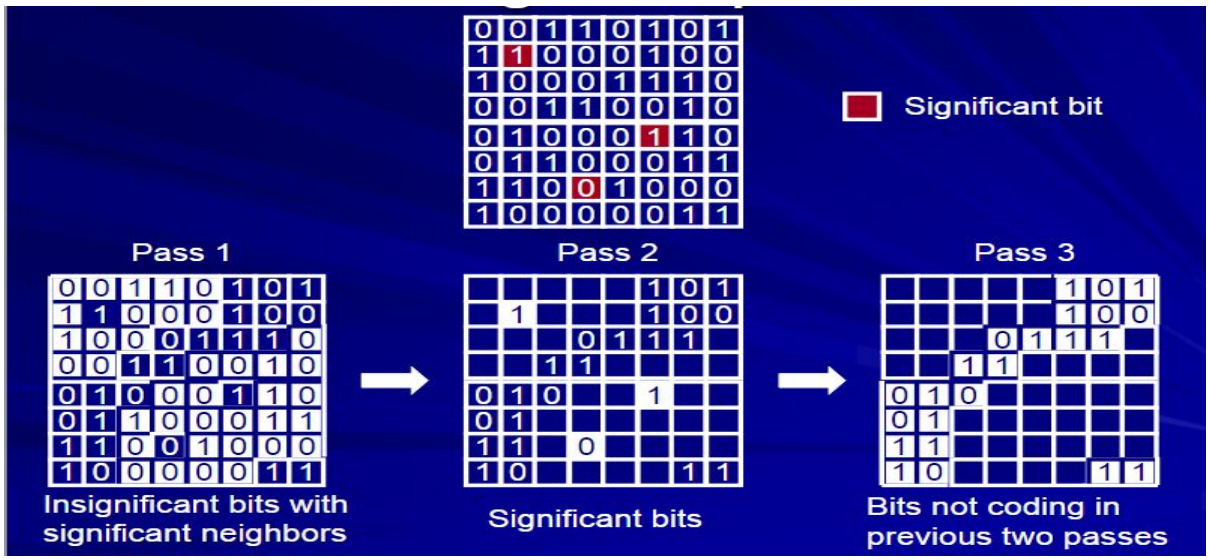
Cleanup pass: Ten etap odnosi się do bitów które zaczynają cały cykl Entropy Decoder (pierwszy bit plane). Ponadto, obsługiwane tutaj są wszystkie bity bit plane'a, które nie zostały obsłużone w poniższych etapach.

Significance propagation decoding pass: etap,, który na podstawie wektora ważności 8 sąsiadów ustala kontekst przekazywany do arytmetycznego dekodera. Wykorzystuje mapowanie zawarte w dokumentacji (D.3.1)

Sign bit decoding - etap, który na podstawie wektora ważności sąsiadów i ich znaków ustala znak współczynnika.

Magnitude refinement pass: ten etap obejmuje tylko te bity współczynników, które już zostały ważne (oprócz tych, które status "ważności" uzyskały dopiero w ostatniej significance propagation pass).

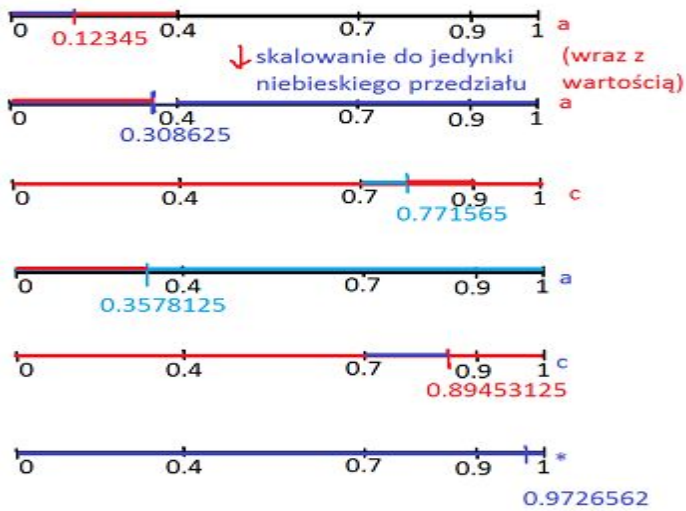
Przykład:



5. Arytmetyczne dekodowanie:
Szczegóły algorytmu arytmetycznego dekodera JPEG-a 2000 można sprawdzić w dokumentacji. Tutaj przedstawiamy pewną intuicję dotyczącą tego zagadnienia.
Punktem wyjścia każdego algorytmu opierającego się na entropii są znane prawdopodobieństwa wystąpienia każdego z symboli. Dekoder dostaje na wejściu specyfikację (mapę) symbol:prawdopodobieństwo. Przyjrzyjmy się przykładowi, który pokazuje istotę algorytmu:
Dany jest alfabet = {a,b,c,*} (* - symbol końca słowa). Każdemu z symboli przyporządkowano prawdopodobieństwa, które odczytujemy ze słownika: {a:0.4, b:0.3,c:0.2, *:0.1 } Zdekodować sekwencję symboli tego alfabetu, mając na wejściu 0.12345.
Jak widać, prawdopodobieństwa sumują się do 1. Tworzymy kolejne podprzedziały przedziału [0,1] o długościach odpowiadającym prawdopodobieństwom kolejnych symboli - tak jak poniżej na rysunku. Wejściowy kod “dopasowujemy” do przedziału - zwracamy etykietę/symbol danego podprzedziału, do którego należy kod. Czynność powtarzamy na przeskalowanym do 1 przedziałowi, od początku podprzedziału, do wartości kodu. Sam kod skalujemy przez długość pierwotnego podprzedziału (od początku do końca). Koniec następuje, gdy kod wpadnie w przedział odpowiadający symbolowi końca (*).
Ciąg kroków w postaci tabelki:

przeskalowany kod	podprzedział	symbol
0.12345	[0; 0.4)	a
0.12345/0.4 = 0.308625	[0; 0.4)	a
0.30865 / 0.4 = 0.771565	[0.7; 0.9)	c
(0.771565 - 0.7)/ 0.2 = 0.3578125	[0; 0.4)	a
0.3578125/0.4 = 0.89453125	[0.7; 0.9)	c
(0.89453125 -.07)/0.2 = 0.9726562	[0.9; 1)	*

Ciąg kroków w postaci graficznej:



Zachowanie algorytmu można uogólnić na alfabet binarny. Zamiast przedziałów różnych długości posługujemy się przedziałami o długości 0.5. “Przełączenie” pomiędzy przedziałami

dokonuje się, gdy któreś skalowanie da inny przedział niż dotychczas. Zwracane są nie symbole a,b,c, lecz 0,1.
W istocie tak właśnie wygląda dekodowanie arytmetyczne w JPEG-2000. Prawdopodobieństwa są wyspecyfikowane w specjalnej tabelce Qe podanej w dokumentacji. Standard podaje wartości, przy których należy “przełączyć” daną wartość na inną. Kontekst dostarczany do arytmetycznego dekodera specyfikuje, których prawdopodobieństw należy użyć.

6. Przykładowe dekodowanie:

```
00000  FF4F  FF51  0029  0000  0000  0001  0000  0009
00020  0000  0000  0000  0000  0000  0001  0000  0009
00040  0000  0000  0000  0000  0001  0701  01FF  5C00
00060  0740  4048  4850  FF52  000C  0000  0001  0001
00100  0404  0001  FF90  000A  0000  0000  001E  0001
00120  FF93  C7d4  0C01  8F0D  C875  5DC0  7C21  800F
00140  B176  FFD9
```

Pierwsza kolumna pokazuje offsety (w bajtach) od początku pliku. kolumny 1-10 to właściwe bajty pliku formatu .jp2.
Pierwszych 120 bajtów to treść metadanych. Interesuje nas to, co znajduje się po bajcie nr 122 (po markerze 0xff93 oznaczającym początek zakodowanych danych).Na wejście PacketDecoder idzie następujący strumień:

Table J.20 – Decoding first packet header

Codestream bytes	Bits used	Comments
0xC7	1	Nonzero length packet
	1	Only code-block is included
	0001	3 zero bit-planes[a)]
0xD4	11 1101010	16 coding passes for this code-block[b)]
	0	*LBlock* remains 3
0x0C	0000110	6 bytes of compressed data[c)]
	0	unused padding bit

a) Maximum bit-planes from Equation (E-2) is 9, 3 of them are zero, so first bit will be decoded into bit-plane 6.

b) Number of bits to read depends on compressed data; see Table B.2.

c) Number of bits to read is *LBlock* plus floor of base 2 log of number of passes: $3 + \lfloor \log 16 \rfloor = 7$.

Wykorzystując informację, że długość pierwszego bloku wynosi 6 bajtów, PacketDecoder żąda od Subbandu wskaźnik na obiekt CodeBlock. Wypełniony zostaje bufor *data tego obiektu bajtami 0x01, 0x8f, 0x0d, 0xc8, 0x75, 0x5d.

Następnie swoje akcje podejmuje EntropyDecoder. Tabelka poniżej pokazuje podejmowane przez niego decyzje, konteksty wprowadzane na wejście arytmetycznego dekodera, a także zdekodowane symbole.

Table J.22 – Arithmetic decode of first code-block

CTX	Context Type	Bit	Comment
17	C4(ZERO_RUN)	1	No zero run.
18	C5(UNIFORM)	1	First nonzero coefficient is the fourth (numbered from 1).
18	C5(UNIFORM)	1	
9	C2(SIGN)	1	Negative.
3	C1 (NEW_SIGNIFICANT)	0	Fifth coefficient is not significant.
3	C1 (NEW_SIGNIFICANT)	1	Third coefficient is significant (first coefficient which is in the significance pass).
10	C2 (SIGN)	0	Negative (XOR bit is 1).
3	C1 (NEW_SIGNIFICANT)	1	Fifth coefficient significant in this coding pass.
10	C2 (SIGN)	0	Negative (XOR bit is 1)
15	C3 (REFINE)	0	Next bit of fourth coefficient is 0.
0	C1 (NEW_SIGNIFICANT)	1	First coefficient is significant.
9	C2 (SIGN)	1	Negative.
4	C1 (NEW_SIGNIFICANT)	1	Second coefficient is significant.
10	C2 (SIGN)	0	Negative.
15	C3 (REFINE)	1	All coefficients are in the refinement pass. Decoded bit is the next bit of the coefficient in order from first to fifth.
15	C3 (REFINE)	0	
15	C3 (REFINE)	1	
16	C3 (REFINE)	0	
15	C3 (REFINE)	0	
16	C3 (REFINE)	0	Next bit-plane.
16	C3 (REFINE)	1	
16	C3 (REFINE)	1	
16	C3 (REFINE)	0	
16	C3 (REFINE)	0	
16	C3 (REFINE)	1	Next bit-plane.
16	C3 (REFINE)	1	
16	C3 (REFINE)	1	
16	C3 (REFINE)	0	
16	C3 (REFINE)	1	
16	C3 (REFINE)	0	Last bit-plane.
16	C3 (REFINE)	0	
16	C3 (REFINE)	0	
16	C3 (REFINE)	0	
16	C3 (REFINE)	1	

Zdekodowane współczynniki WT wyglądają następująco: -26, -22, -30, -32, -19.
Podobnie dekodowany jest drugi blok w tym przykładzie.

7. Bibliografia:

1. ITU-T T.800: Information technology – JPEG 2000 image coding system: Core coding system
2. http://www.whydomath.org/node/wavlets/hwt.html

Część trzecia - JPG LS