Damian Karwowski

ZROZUMIEĆ Kompresję Obrazu

Podstawy Technik Kodowania Stratnego oraz Bezstratnego Obrazów

Wydanie Pierwsze
Poznań 2019

Damian Karwowski

Zrozumieć
Kompresję Obrazu

Podstawy Technik Kodowania Stratnego oraz Bezstratnego Obrazów

Wydanie Pierwsze, wersja 1.3

Poznań 2019r.

ISBN 978-83-953420-0-4

9 788395 342004

Egzemplarz książki został zdeponowany w Kancelarii Notarialnej.

Książka jest dostępna pod adresem: www.zrozumieckompresje.pl

ISBN 978-83-953420-0-4

Projekt okładki książki oraz strona internetowa zostały wykonane przez Marka Piskulskiego. Serdecznie dziękuję za profesjonalną pracę ☺

Obraz „Miś Panda”, który jest częścią okładki, namalowała na płótnie Natalka Karwowska. Natalko, dziękuję Ci za Twój wysiłek ☺

Autor książki

Dr inż. Damian Karwowski. Absolwent Politechniki Poznańskiej. Uzyskał tytuł zawodowy **magistra inżyniera** oraz stopień **doktora** nauk technicznych na Politechnice Poznańskiej, odpowiednio w latach 2003 i 2008. Obecnie pracownik **naukowo-dydaktyczny** na wyżej wymienionej uczelni. Od roku 2003 zawodowo zajmuje się **kompresją obrazu**. Autor ponad 50 publikacji naukowych o tematyce **kompresji i przetwarzania obrazów**. Brał udział w licznych **projektach naukowych** dotyczących wydajnej reprezentacji multimedialnych danych. Dodatkowo członek wielu zespołów badawczych, które w tematyce **kompresji obrazu i dźwięku** realizowały **prace badawczo-wdrożeniowe** dla przemysłu. Jego zainteresowania obejmują kompresję danych, techniki kodowania entropijnego danych oraz realizację kodeków obrazu i dźwięku na procesorach **x86** oraz **DSP**.

Spis treści

Spis treści 5

“Słowo” wstępu 11

Rozdział 1 **15**

Obraz i jego reprezentacja przestrzenna 15

1.1. Wprowadzenie 15
1.2. Próbkowanie obrazu – podstawy podstaw 17
1.3. Kwantyzacja obrazu 20
1.3.1. Kwantyzacja danych – podstawowe informacje 20
1.3.2. Kwantowanie próbek obrazu monochromatycznego 22
1.4. Reprezentacje cyfrowego obrazu kolorowego 24
1.4.1. Krótkie wprowadzenie 24
1.4.2. Obraz kolorowy w przestrzeni RGB 25
1.4.3. Obraz kolorowy w przestrzeni RGB – ograniczenia reprezentacji barw 27
1.4.4. Przestrzeń RGB – odmiany i zastosowanie 29
1.4.5. Obraz kolorowy w przestrzeni YC_BC_R 31
1.4.6. Kompresja kolorowego obrazu – dlaczego właśnie w przestrzeni YC_BC_R? 33
1.4.7. Przestrzeń YC_BC_R i schematy próbkowania chrominancji 33
1.4.8. Schematy próbkowania chrominancji – jaki jest sens przyjętych oznaczeń liczbowych? 36
1.4.9. Powrót z reprezentacji YC_BC_R do przestrzeni RGB 37
1.5. Obraz ruchomy, czyli cyfrowa sekwencja wizyjna 37
1.6. Podsumowanie 39

Rozdział 2 **41**

Częstotliwościowa reprezentacja obrazu 41

2.1. Wprowadzenie 41
2.2. Obraz i jego reprezentacja częstotliwościowa – podstawy 42
2.2.1. Zmiany treści w kierunku poziomym, czyli „częstotliwość przestrzenna pozioma” 43
2.2.2. Zmiany treści w kierunku pionowym, czyli „częstotliwość przestrzenna pionowa” 45
2.2.3. Bardziej rzeczywisty przypadek, czyli zmiany w obu kierunkach naraz 46
2.3. Częstotliwościowa „zawartość” (reprezentacja) obrazu naturalnego 47
2.3.1. Obraz i jego widmo amplitudowe 47
2.3.2. Widmo fazowe obrazu 50
2.4. Reprezentacja „częstotliwościowa” obrazu w ujęciu matematycznym 51
2.5. Dyskretne przekształcenie Fouriera – dalszy komentarz 54
2.6. Praktyczne użycie dyskretnego przekształcenia Fouriera – istotny problem 56
2.7. Eliminacja problemu „skokowej” zmiany wartości sygnału na granicach cykli 57
2.8. DFT „nowego sygnału okresowego” – czyli w efekcie dyskretne przekształcenie kosinusowe (DCT) 59
2.8.1. DCT sygnału jednowymiarowego, czyli 1D-DCT 59
2.8.1.1. Wyprowadzenie matematyczne 59
2.8.1.2. „Wygląd” kosinusoid przekształcenia 1D-DCT 63
2.8.1.3. Ilustracja związku DCT z DFT 65
2.8.2. DCT sygnału dwuwymiarowego, czyli 2D-DCT 65

2.8.3. Dalsza interpretacja oraz znaczenie 2D-DCT w praktyce kompresji obrazów 67
2.8.3.1. „Wygląd” kosinusoid przekształcenia 2D-DCT 67
2.8.3.2. Interpretacja rezultatu 2D-DCT raz jeszcze 68
2.8.3.3. Na czym polega „siła” 2D-DCT? 70
2.8.4. Inny sposób matematycznego opisu przekształcenia DCT 72
2.8.5. Dlaczego nie przekształcenie sinusowe zamiast kosinusowego? 75
2.8.6. Sposób użycia 2D-DCT w koderach obrazu 79
2.9. Inne stosowane przekształcenie – transformacja Walsha-Hadamarda 82
2.10. Przekształcenie „najlepsze z najlepszych”, czyli transformacja Karhunena-Loévego 84
2.11. Reprezentacja obrazu w dziedzinie częstotliwości – dlaczego warto? 85
2.12. Podsumowanie 89

Rozdział 3 **91**

Kodowanie entropijne 91

3.1. Wprowadzenie 91
3.2. Kodowanie o zmiennej długości słowa – VLC 93
3.2.1. Kodowanie Huffmana 93
3.2.1.1. Kodowanie Huffmana – szczegóły algorytmu podstawowego 93
3.2.1.2. Efektywność podstawowej wersji kodowania Huffmana 95
3.2.2. Blokowe kodowanie Huffmana 95
3.2.3. Efektywność przedstawionych technik kodowania Huffmana w praktyce kompresji danych multimedialnych – krótki komentarz 96
3.2.4. Dynamiczne kodowanie Huffmana 97
3.2.4.1. Opis ogólny 97
3.2.4.2. Opis szczegółowy – drzewo kodów Huffmana 98
3.2.4.3. Opis szczegółowy – uaktualnienie drzewa kodów Huffmana 98
3.2.5. „Uniwersalne” kodowanie o zmiennej długości słowa kodowego 104
3.2.5.1. Kodowanie unarne 105
3.2.5.2. Kodowanie Golomba 106
3.2.5.3. Kodowanie Rice’a (Golomba-Rice’a) 112
3.2.5.4. Kodowanie Exp-Golomba *k*-tego rzędu 112
3.3. Kodowanie arytmetyczne 116
3.3.1. Wprowadzenie 116
3.3.2. Kodowanie arytmetyczne – algorytm podstawowy 117
3.3.3. Kodowanie arytmetyczne – algorytm podstawowy w ujęciu matematycznym 118
3.3.4. Dekodowanie arytmetyczne danych – krótka dyskusja 120
3.3.5. Efektywność techniki kodowania arytmetycznego 120
3.3.6. Algorytm podstawowy kodowania arytmetycznego – dwa istotne problemy 121
3.3.7. „Prawie” praktyczna realizacja kodowania arytmetycznego 122
3.3.8. Problem przekroczenia precyzji rejestrów w kodeku 124
3.3.9. „Całkowitoliczbowa” realizacja kodowania arytmetycznego – końcowy algorytm 126
3.3.10. „Całkowitoliczbowa” realizacja dekodowania arytmetycznego 126
3.3.11. Algorytm kodowania arytmetycznego w zapisie dwójkowym 128
3.3.12. Szybkie realizacje kodowania arytmetycznego 128
3.3.13. Modelowanie statystyczne danych w kodowaniu arytmetycznym – krótka dygresja 129
3.4. Efektywność kompresji oraz złożoność współczesnych technik kodowania entropijnego 131
3.4.1. Efektywność kompresji algorytmów CABAC i UVLC 131

3.4.2. Złożoność obliczeniowa technik CABAC i UVLC 133
3.5. Podsumowanie 134

Rozdział 4 135

Wybrane techniki kompresji obrazu 135

4.1. Wprowadzenie 135

Część I – Kompresja statycznego obrazu 138

4.2. Obraz kolorowy w przestrzeni YC_BC_R – czyli wstęp do kompresji 138
4.3. Statystyczna nadmiarowość próbek obrazu – prosty sposób jej redukcji 140
4.4. Przewidywanie danych obrazu jako sposób na poprawę wyników kompresji 142
4.5. Przewidywanie wartości próbek obrazu – jak to się robi w praktyce? 145
4.6. Nadmiarowość przestrzenna w obrazie – inny sposób jej redukcji 152
4.7. Jeszcze lepsze rozwiązanie, czyli łączne użycie metody przewidywania wartości próbek z opisem wyniku funkcjami kosinusoidalnymi 154
4.8. Usunięcie części informacji o obrazie jako sposób na istotne zwiększenie stopnia kompresji danych 157
4.9. Kompresja obrazu w dziedzinie częstotliwości – furtka do wydajnej realizacji idei kodowania stratnego 159
4.10. Praktyczna realizacja techniki kodowania transformatowego 163
4.10.1. Kodowanie transformatowe w bardzo prostym wydaniu 163
4.10.1.1. Podział obrazu na małe fragmenty (bloki) 164
4.10.1.2. Reprezentacja treści bloków w dziedzinie częstotliwości 165
4.10.1.3. Kwantyzacja składowych kosinusoidalnych 167
4.10.1.4. Szeregowanie skwantowanych kosinusów 171
4.10.1.5. Kodowanie skwantowanych amplitud kosinusów 172
4.10.1.6. Kodowanie entropijne danych 174
4.10.1.7. Wydajność prostych metod kompresji transformatowej 174
4.10.1.7.1. Efektywność kompresji 174
4.10.1.7.2. Złożoność obliczeniowa 178
4.10.2. Kodowanie transformatowe w wydaniu zaawansowanym 179
4.10.2.1. Przewidywanie treści obrazu 181
4.10.2.2. Kodowanie transformatowe błędu przewidywania treści bloków 186
4.10.2.3. Kodowanie skwantowanych próbek widma 187
4.10.2.4. Filtracja obrazu poprawiająca jakość 188
4.10.2.4.1. Efekt blokowy i sposób jego redukcji 189
4.10.2.4.2. Efekt dzwonienia i sposób jego redukcji 193
4.10.2.5. Wydajność zaawansowanych technik kompresji transformatowej 195
4.10.2.5.1. Efektywność kompresji 195
4.10.2.5.2. Złożoność obliczeniowa 199

Część II – Kompresja ruchomego obrazu 200

4.11. Kodowanie ruchomego obrazu – najprostsze podejście 200
4.12. Nadmiarowość czasowa sekwencji – cenna wskazówka dla jeszcze wydajniejszej kompresji 200
4.13. Jak wydajnie przewidywać obrazy sceny, która zmienia się w czasie? 202
4.14. Estymacja ruchu w ujęciu matematycznym 204
4.15. Jaka jest skuteczność międzyobrazowej, jednokierunkowej predykcji treści? 204
4.16. Międzyobrazowa predykcja treści – przewidywanie nie tylko z przeszłości, ale i przyszłości 207
4.17. Predykcja międzyobrazowa jedno- i dwukierunkowa. Która jest lepsza? 209

4.18. Sposób na dodatkowe zwiększenie dokładności metody – estymacja ruchu z ułamkową dokładnością 212
4.18.1. Wprowadzenie teoretyczne 212
4.18.2. Praktyczna realizacja estymacji ruchu z ułamkową dokładnością 217
4.18.3. Jak dokładniejsza estymacja ruchu wpływa na efektywność kodera obrazu? 218
4.19. Standardowy sposób estymacji ruchu – złożoność obliczeniowa 218
4.20. Szybkie metody estymacji ruchu 220
4.21. Szybkie metody estymacji ruchu – ocena skuteczności i złożoności obliczeniowej 224
4.22. Międzyobrazowa predykcja treści obrazu. W jak dużych blokach najlepiej ją stosować? 226
4.23. Jeszcze o praktycznej realizacji estymacji ruchu – bardzo krótkie podsumowanie 228
4.24. Estymacja ruchu metodą pasowania bloków. Jak ocenić stopień podobieństwa dwóch bloków obrazu? 229
4.25. Estymacja ruchu metodą pasowania bloków. Czy ta metoda trafnie wyznacza ruch obiektów sekwencji? 231
4.26. Kodowanie transformatowe błędu przewidywania treści bloków 232
4.27. Kompletny koder obrazu, czyli łączne użycie wielu narzędzi kodowania danych 233
4.28. Jeszcze o filtracji, która poprawia jakość obrazów 236
4.29. Pętla sprzężenia zwrotnego w koderze obrazu 237
4.30. Poziomy wyboru wariantów kompresji, czyli typy obrazów i typy bloków 238
4.30.1. Poziom bloku wyboru wariantu kompresji 238
4.30.2. Poziom obrazu wyboru wariantów kompresji 241
4.31. Jak kodowana sekwencja dzielona jest na obrazy określonego typu? 241
4.32. Efektywność kompresji obrazów poszczególnych typów 244
4.33. Złożoność kodowania i dekodowania obrazów 245
4.34. Kolejność kodowania a kolejność wyświetlania obrazów sekwencji 248
4.35. Złożoność hybrydowego kodera i dekodera obrazów 249

Część III – Sterowanie koderem obrazu 250

4.36. Wprowadzenie 250
4.37. Jakie są najważniejsze cele sterowania koderem obrazu? 250
4.38. W jaki sposób decydować o trybach kodowania bloków oraz sile kwantowania danych? 251
4.39. Przykłady wybranych wariantów sterowania koderem obrazu 254
4.39.1. Kodowanie ze stałym Qs 254
4.39.2. Kodowanie w wariancie CBR 255
4.39.3. Kodowanie w wariancie ABR 255
4.39.4. Kodowanie VBR 255
4.39.5. Wariant sterowania z optymalizacją R-D 257
4.39.5.1. Krótkie wprowadzenie 257
4.39.5.2. Optymalizacja R-D w ujęciu matematycznym 257

Część IV – Rozwój technik kompresji obrazu z perspektywy ostatnich 30 lat 259

4.40. Wprowadzenie 259
4.41. Ewolucja techniki kodowania hybrydowego 260
4.41.1. Podział obrazu na bloki 260
4.41.2. Liczba dostępnych trybów kompresji 261
4.41.3. Poprawa jakości obrazów po kompresji 264
4.42. Ewolucja efektywności kompresji obrazów 266

4.43. Jak zmieniła się złożoność kodowania obrazów? 268
4.44. Realizacja oprogramowania kodeka – jak zmieniła się trudność? 269
4.45. Jaka jest najbliższa przyszłość? 270
4.46. Podsumowanie rozdziału i wnioski końcowe 274

Bibliografia 277

“Słowo” wstępu

Obecnie przeszło **70%** wszystkich danych, jakie są przesyłane w sieciach teleinformatycznych, to dane, które reprezentują **obraz**. Nam widzom, obraz kojarzy się z rozrywką. Są to zdjęcia, filmy i filmiki, obrazy ze świata gier komputerowych, a więc wszystko to, co jest dla przeciętnego widza bardzo atrakcyjne. Dlatego prognozuje się, że w ciągu kilku najbliższych lat udział obrazu w sieciach jeszcze się zwiększy i już w 2021 roku przekroczy **80%**.

Zdjęcia czy filmy są więc dla nas odbiorców szczególnym typem danych. Tak szerokie wykorzystanie obrazu zawdzięczamy coraz łatwiejszemu dostępowi do aparatów i kamer, rozwojowi wyświetlaczy, prężnemu rozwojowi infrastruktury sieciowej (potrafimy przesyłać coraz więcej danych w tym samym czasie), czy wreszcie postępowi w zakresie urządzeń magazynowania danych (na nośnikach pamięci potrafimy zapisać coraz więcej informacji).

Ale jest coś jeszcze, bez czego wykorzystanie obrazu na tak masową skalę, nie byłoby na pewno możliwe. Tym czymś jest **kompresja danych obrazu**, czyli zadanie przedstawienia treści obrazu przy pomocy możliwie małej liczby bitów. Oryginalne, nieskompresowane dane, które prezentują, na przykład film full HD (przy założeniu 30 obrazów na sekundę), to gigantyczny strumień bitów o wielkości **1,5 Gigabitów/s !** Jeśli ktoś ma w swoim domu łącze internetowe o szybkości **100 Megabitów/s**, to potrzebowałby aż **15** takich łączy, żeby móc taki film oglądać na żywo przez Internet! Jest zatem oczywiste, że operowanie na oryginalnych danych obrazu nastręczałoby olbrzymich trudności lub byłoby technicznie niemożliwe. W pierwszym z wymienionych przypadków problemem na pewno byłyby ogromne koszty przetwarzania i transmisji danych.

Z perspektywy wielu zastosowań **kompresja obrazu** jest więc koniecznością. I jej znaczenie jest rzeczywiście ogromne. Pozwala ona przedstawić treść obrazu w inny, alternatywny sposób, który w stosunku do reprezentacji oryginalnej zużywa na ten cel nieporównywalnie mniejszą liczbę bitów. To trochę tak, jakbyśmy potężny zbiór danych oryginalnych obrazu mocno ścisnęli imadłem, z każdej strony, powodując silną redukcję „objętości” danych. Ten wynikowy, pomniejszony strumień danych znacznie łatwiej jest przesyłać w Internecie, w systemach telewizji cyfrowej, czy zapisywać na dysku komputera. Koszty operowania na zredukowanym zbiorze będą oczywiście nieporównywalnie mniejsze. Dlatego kompresja danych ma tak wielkie znaczenie w multimedialnym świecie.

Z tego powodu zagadnienie wydajnej reprezentacji obrazu jest przedmiotem szczególnego zainteresowania inżynierów i naukowców, i to co najmniej od lat 70-tych poprzedniego stulecia. Oprócz wysiłków dużej liczby uniwersytetów, badania nad kompresją są także od lat prowadzone w licznych laboratoriach światowych koncernów branży IT. I to dużych koncernów, jak np. Microsoft, Google czy Samsung. W przypadku wymienionych podmiotów (uczelnie, firmy) kompresja obrazu stanowi nie tylko atrakcyjny przedmiot badań i prac wdrożeniowych, ale otwiera również rynki zbytu dla produktów i usług w bardzo szeroko pojętym sektorze multimediów. Ogromne zainteresowanie tematem kompresji, nie może więc dziwić.

Dobre zrozumienie fundamentów kompresji jest zatem bardzo potrzebne, tak z perspektywy prowadzenia badań naukowych, jak i wszelkich prac rozwojowo-wdrożeniowych. Wiedzą na temat kompresji są więc żywo zainteresowane nie tylko jednostki, które zajmują się kształceniem i prowadzeniem prac naukowych, ale również komercyjne firmy. Dlatego dzisiaj zagadnienia kompresji danych, w tym kompresji obrazu, są ważnym elementem programu kształcenia studentów na wielu kierunkach studiów wyższych, uczelni polskich, jak i zagranicznych. W Polsce w głównej mierze dotyczy to wyższych uczelni technicznych. Dobrym tego przykładem

są wybrane kierunki studiów na Politechnikach: Elektronika i Telekomunikacja, Teleinformatyka, Automatyka i Robotyka, Informatyka. Niektóre firmy, jak chociażby dostawcy sygnału telewizyjnego, świadczą usługi oraz realizują wdrożenia, które również wymagają od ich pracowników dobrego rozumienia technik kompresji obrazu.

Z powyższej perspektywy mogłoby się zatem wydawać, że na temat kompresji obrazu, dostępnych jest na polskim rynku wiele różnych książek i publikacji. Opracowań, które zostały przygotowane w języku polskim. Jednak w rzeczywistości jest niestety odwrotnie. Popularnonaukowych publikacji, które prezentowałyby podstawy i założenia metod kompresji prawie nie ma. Podobnie jest w przypadku książek i podręczników. Tych jest dosłownie kilka tytułów, przy czym, tylko w niektórych z tych kilku mowa jest o zagadnieniach kompresji danych, które reprezentują obrazy. Dlatego zdecydowałem się napisać tę książkę. Moja książka w całości jest poświęcona tym algorytmom kompresji danych, które znajdują szerokie, praktyczne zastosowanie w technikach **kompresji obrazów statycznych** (czyli zdjęć) i **sekwencji wizyjnych** (czyli filmów).

Kompresja obrazu, jako zagadnienie techniczne, nie jest na pewno tematem łatwym. Jednak można o niej mówić prostym i przystępnym dla odbiorcy językiem. Jako nauczyciel akademicki w taki właśnie sposób staram się mówić o kompresji podczas zajęć dydaktycznych, które na co dzień prowadzę ze studentami. Taki sam cel postawiłem również przed tą książką. Prostota przekazu, bez przesadnego teoretyzowania. Dlatego, zamiast już na starcie „atakować” i „przerażać” Czytelnika gąszczem matematycznych wzorów, czy naukowych wywodów, które same w sobie na pewno nie ułatwiają zrozumienia omawianego zagadnienia, w tej książce staram się prezentować materiał według idei „od ogółu, do szczegółu”. Mówiąc przy tym łatwym do zrozumienia językiem. Najpierw istota każdego problemu jest więc przedstawiana „z lotu ptaka”, a dopiero później, w kolejnych punktach książki, Czytelnik zapoznaje się z bardziej szczegółowymi informacjami na dany temat. Poszczególne fragmenty tekstu są przy tym uzupełnione licznymi ilustracjami, wykresami i schematami, żeby uczynić przekaz jeszcze bardziej czytelnym. Aparat matematyczny, który ma zastosowanie w danej technice, jest oczywiście też ważny. Jednak w tej książce znajduje się on na końcu łańcucha prezentacji, co ma ułatwić lepsze jego zrozumienie u Czytelnika. Książka ta jest więc bardziej popularnonaukowym opracowaniem niż czysto naukowym podręcznikiem. Jej odbiorcami mogą być zatem nie tylko osoby, które już coś wiedzą na temat kompresji obrazu, ale również zupełnie początkujący Czytelnicy, w tym **studenci**. Mam nadzieję, że przyjęta w książce koncepcja prezentacji wiedzy okaże się dla Czytelnika korzystna i atrakcyjna. Że zostanie odebrana, jako główna zaleta tej książki. Bardziej zaawansowany Czytelnik znajdzie w książce wiele odniesień literaturowych dla omawianych zagadnień, co ma stanowić pomoc w dalszym poszerzaniu wiedzy na temat kompresji danych.

Książka prezentuje te spośród metod **kompresji obrazu**, które znajdują praktyczne zastosowanie w używanych na rynku kodekach obrazu, czyli urządzeniach bądź komputerowych programach, które zajmują się kompresją i dekompresją obrazu. W książce uwzględniono również najnowsze metody kompresji, które zostały opracowane już po 2010 roku. W przedstawianym opisie staram się nie poprzestawać na samej tylko teorii metod. Szczególną uwagę staram się dodatkowo zwracać na element właściwego ich zrozumienia. Stąd zaproponowany tytuł książki. Wobec tego Czytelnik dowie się z tego podręcznika nie tylko jak, w teorii, działa dana technika kompresji, ale zdobędzie ponadto praktyczne wiadomości odnośnie motywacji i zasadności stosowania opisanych rozwiązań, ich złożoności obliczeniowej, czy trudności ich implementacji w koderze i dekoderze obrazu. Duża część tej dodatkowej wiedzy jest wynikiem moich osobistych doświadczeń, które miałem okazję zdobyć podczas wieloletniej pracy w charakterze badacza i programisty kodeków. Będę bardzo szczęśliwy, jeśli ta dodatkowa porcja informacji okaże się dla Czytelnika wiedzą wartościową, i że stanie się jednym z atutów tej książki.

Na zakończenie wstępu pragnę życzyć każdemu Czytelnikowi tego, żeby znalazł w książce choćby jeden niewielki fragment, który będzie dla niego cennym źródłem wiedzy. Jeśli tak się stanie, to będzie to dla mnie ogromna satysfakcja. Bardzo bym chciał, żeby ta książka była dla Czytelników dobrym uzupełnieniem lektury innych opracowań na temat kompresji danych i obrazu.

Autor

Damian Karwowski

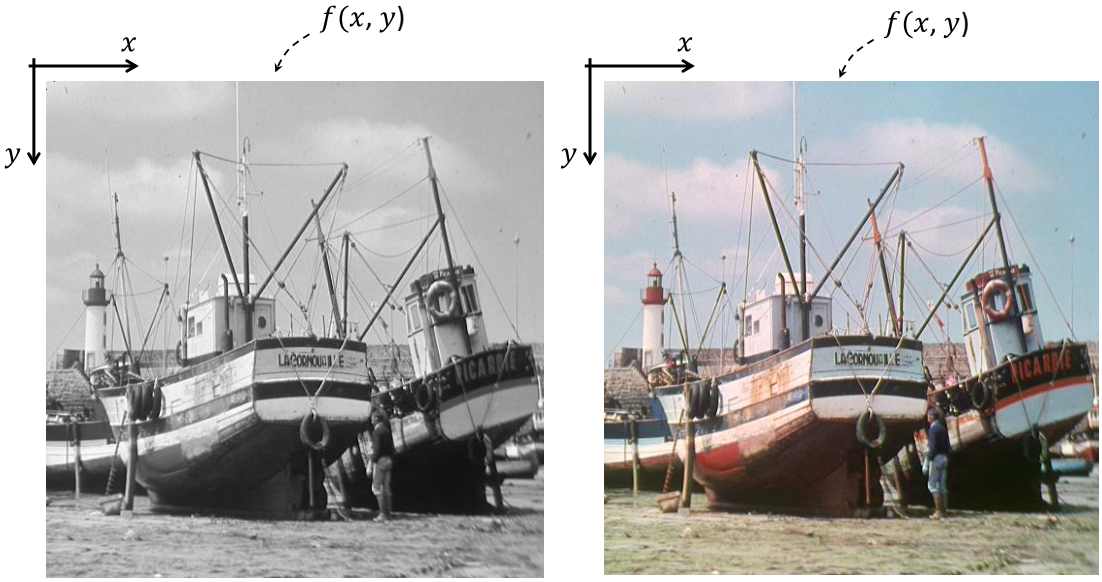
Dla Hani, Natalki i Amelki

Rozdział 1

Obraz i jego reprezentacja przestrzenna

1.1. Wprowadzenie

Zapewne, każdy z Czytelników tej książki dobrze wie, czym jest **obraz**, stąd przytaczanie jego formalnej definicji nie wydaje się być konieczne. Obraz, który jest wyświetlany na klasycznych monitorach, czy telewizorach posiada dwa wymiary: szerokość (mierzona wzdłuż osi x) oraz wysokość (mierzona wzdłuż osi y). Poszczególne fragmenty obrazu odznaczają się pewną **jasnością** (tak jest w przypadku obrazu **jednobarwnego**, inaczej **monochromatycznego**) lub **kolorem** (gdy mowa jest o obrazie **kolorowym**), tak jak na przedstawionych poniżej przykładach.

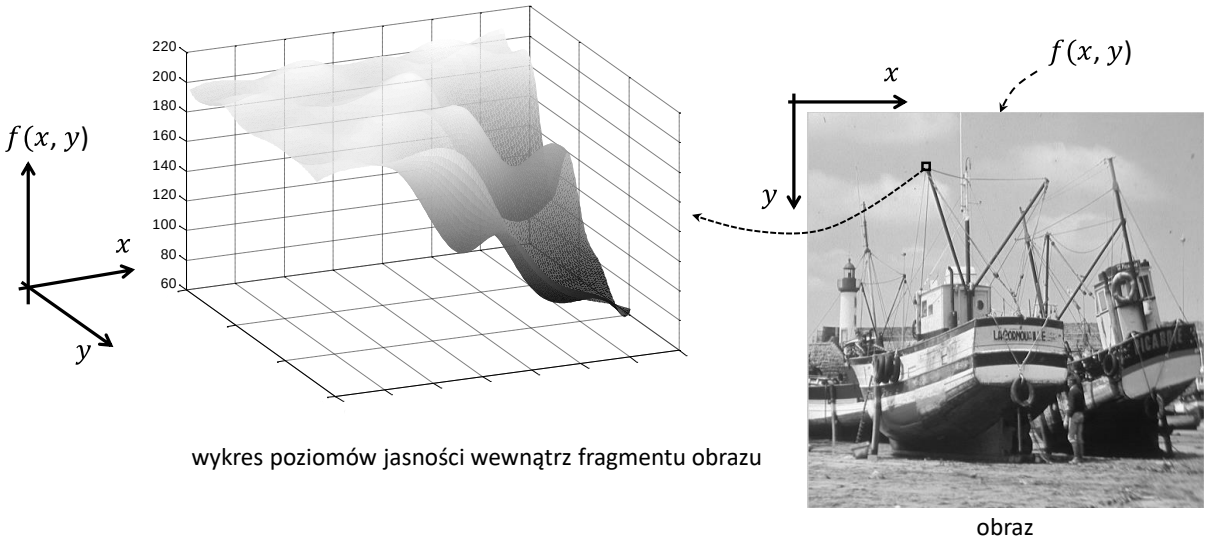


obraz **monochromatyczny** obraz **kolorowy**

Rysunek 1-1 Ilustracja obrazów: monochromatycznego i kolorowego.

Z matematycznego punktu widzenia obraz (monochromatyczny lub kolorowy) jest więc dwuwymiarową funkcją (sygnałem) $f(x, y)$, gdyż wartości tej funkcji zależą od wartości dwóch niezależnych zmiennych: zmiennej x (czyli pozycji w obrazie w kierunku poziomym) i zmiennej y

(pozycji w kierunku pionowym). W przypadku, kiedy x i y będą przyjmować dowolne wartości[1], jak również wartości funkcji $f(x,y)$ będą dowolne (tzn. z zakresu nieujemnych liczb rzeczywistych), wtedy mówimy o obrazie **analogowym**. W ujęciu matematycznym dowolny fragment takiego obrazu jest dwuwymiarową funkcją ciągłą, co w przypadku obrazu monochromatycznego zostało zilustrowane na rysunku 1-2.

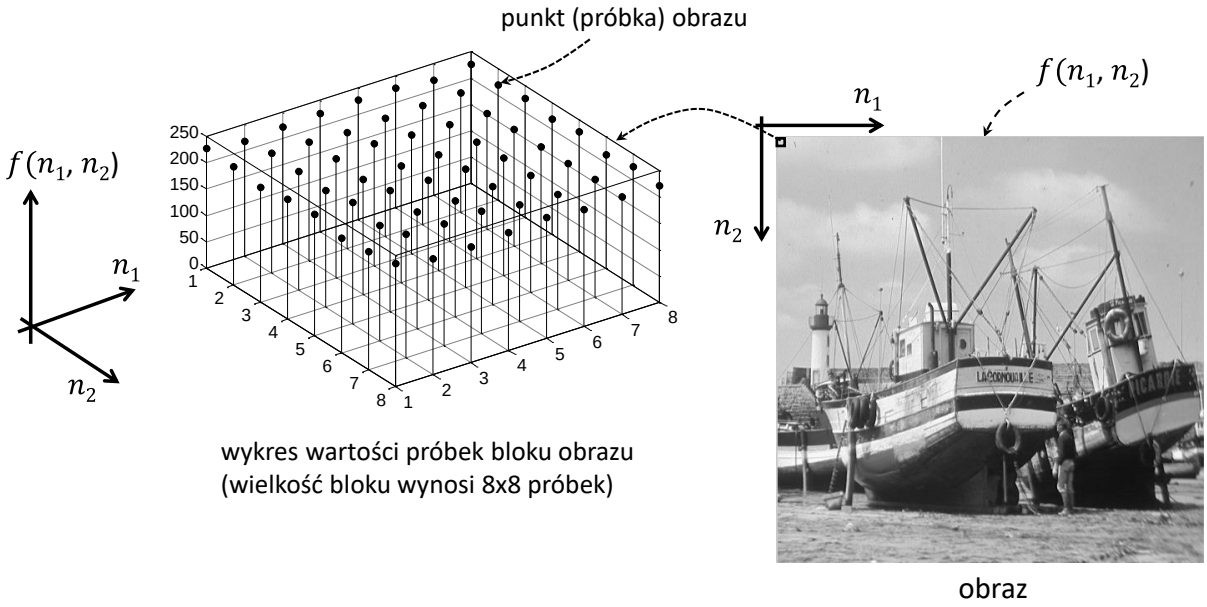


Rysunek 1-2 Interpretacja fragmentu **analogowego obrazu monochromatycznego** jako dwuwymiarowej funkcji ciągłej.

Obraz analogowy pozwala na precyzyjną reprezentację treści (bo w teorii nie ma tutaj żadnych ograniczeń na wartości $f(x,y)$, x i y), jednak stosowanie tego typu obrazu nastręcza wielu różnych trudności. Można powiedzieć, że bardziej problematyczne są wszelkie operacje przetwarzanie i filtracji obrazu, bo nie można do tych celów bezpośrednio zastosować procesorów. Znacznie trudniejszy jest również zapis takiego sygnału. Chociażby z tych wymienionych powyżej powodów, nie operuje się bezpośrednio na obrazie analogowym, ale dokonuje się odpowiedniej jego modyfikacji, celem otrzymania obrazu innego typu. Robi się to w drodze dwuetapowego przetwarzania. I tak, w etapie pierwszym obraz analogowy jest poddawany **próbkowaniu**, czyli ograniczamy wiedzę o wartościach funkcji $f(x,y)$ do pewnych dyskretnych wartości zmiennych x oraz y – oznaczanych odpowiednio jako n_1 oraz n_2. W przeciwieństwie do zmiennych x i y wartości n_1 i n_2 są całkowitoliczbowe. Otrzymany w ten sposób obraz nazywany jest obrazem **dyskretnym**, a pojedynczy element tego obrazu (znajdujący się w obrazie w lokalizacji (n_1, n_2) i mający wartość $f(n_1, n_2)$) **punktem** obrazu lub **próbką** obrazu[2]. W etapie drugim, wartość każdego z punktów obrazu jest poddawana **kwantyzacji** (kwantowaniu), celem zmniejszenia zbioru wszystkich możliwych wartości $f(n_1, n_2)$ dla próbek. Po kwantyzacji wartość każdej z próbek obrazu będzie można zapisać na skończonej, wcześniej ustalonej liczbie bitów (np. na 8 bitach), co szalenie ułatwia „magazynowanie” obrazu (np. na dyskach komputera). Skwantowany obraz dyskretny jest nazywany obrazem **cyfrowym**. Cyfrowy obraz monochromatyczny został zilustrowany na rysunku 1-3.

[1] Oczywiście, w obrębie określonej szerokości i wysokości obrazu.

[2] W niektórych opracowaniach stosuje się również termin piksel dla określenia najmniejszego elementu cyfrowego obrazu. Słowo piksel pochodzi od słów ‘picture element’. Zdaniem części osób terminem tym powinno się określać pojedynczy punkt ekranu wyświetlającego obraz, a nie pojedynczy punkt obrazu zapisanego w pamięci czy na dysku.



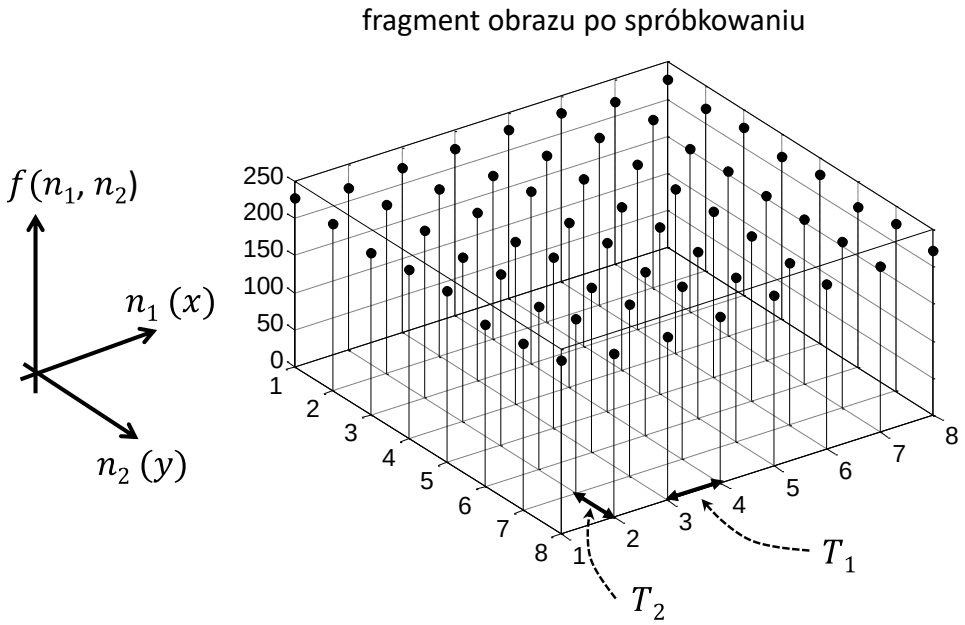
Rysunek 1-3 Interpretacja fragmentu **cyfrowego obrazu monochromatycznego** jako dwuwymiarowej funkcji dyskretnej.

W praktyce, wspomniane próbkowanie oraz kwantowanie obrazu analogowego jest przeprowadzane w aparacie czy kamerze, już na etapie rejestracji obrazu. Sposób, w jaki dokonuje się próbkowania obrazu, ale również sposób kwantowania otrzymanych próbek, mają ogromny wpływ na jakość cyfrowego obrazu. Dlatego zagadnieniom tym zostały poświęcone dwa kolejne punkty książki.

1.2. Próbkowanie obrazu – podstawy podstaw

Istnieją matematyczne zależności mówiące o tym, jak „gęsto” należy próbkować obraz sceny, czyli jakie mogą być maksymalne odległości pomiędzy kolejnymi, pobieranymi próbkami z obrazu sceny, w pionie i w poziomie, żeby zarejestrowany przez aparat czy kamerę obraz był pozbawiony widocznych zniekształceń. Mówi o tym twierdzenie **Kotielnikowa-Shannona** o próbkowaniu sygnałów [Doma10]. Chociaż twierdzenie to jest jednym z fundamentalnych twierdzeń teorii sygnałów, to z uwagi na założony zakres tematyczny tej książki próbkowanie obrazów nie będzie przedmiotem szczegółowych, matematycznych rozważań. Zamiast tego przytoczone zostaną podstawowe związki, pozwalające na prawidłowe zrozumienie istoty procesu próbkowania obrazu.

Ponieważ obraz jest sygnałem dwuwymiarowym, próbki obrazu należy pobierać wzdłuż jego szerokości x oraz wysokości y. W praktyce najczęściej stosuje się tzw. **prostokątne próbkowanie**, które można sobie wyobrazić jako efekt nałożenia na próbkowany obraz $f(x, y)$ prostokątnej siatki, i pobraniu próbek obrazu w miejscach występowania węzłów siatki. Próbki obrazu są pobierane z ustalonym wcześniej **okresem próbkowania** T_1 wzdłuż kierunku x oraz okresem próbkowania T_2 wzdłuż kierunku y w obrazie. Te okresy próbkowania to odległość pomiędzy kolejnymi próbkami spróbkowanego obrazu. Przyjęte wartości T_1 i T_2 przekładają się na **częstotliwości próbkowania** f_{s1} i f_{s2} obrazu, tak jak przedstawiono to na rysunku 1-4.

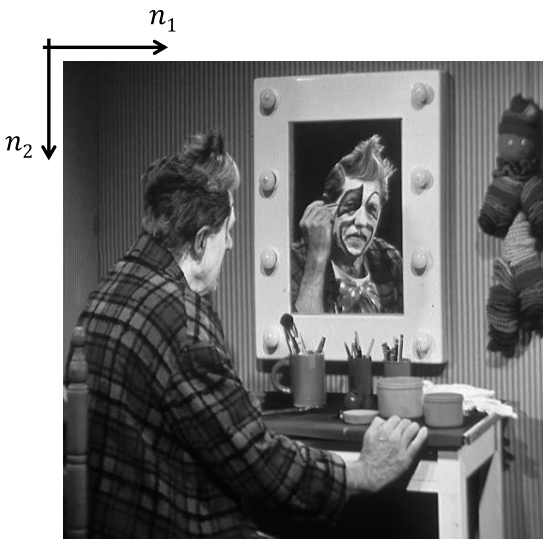


$T_1 -$ okres próbkowania w kierunku $\boldsymbol{x}$
$T_2 -$ okres próbkowania w kierunku $\boldsymbol{y}$

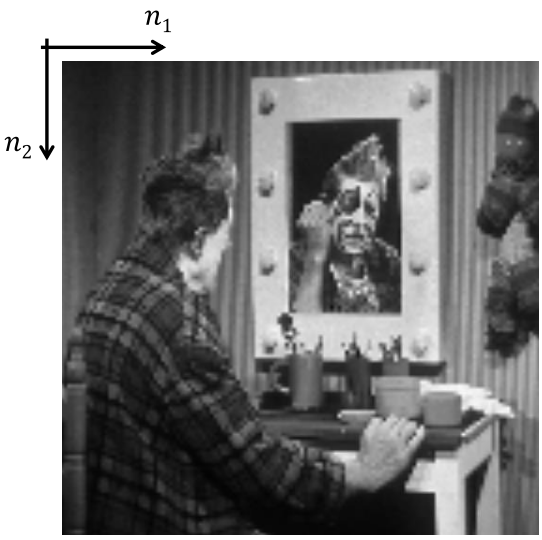
$f_{s1} = 1/T_1 -$ częstotliwość próbkowania w kierunku $\boldsymbol{x}$
$f_{s2} = 1/T_2 -$ częstotliwość próbkowania w kierunku $\boldsymbol{y}$

Rysunek 1-4 Ilustracja próbkowania prostokątnego dla obrazu. Na rysunku zaznaczono okresy próbkowania T_1 i T_2 w kierunkach odpowiednio poziomym (x) oraz pionowym (y). Zastosowany indeks s ma pochodzenie od słowa w języku angielskim „sampling”.

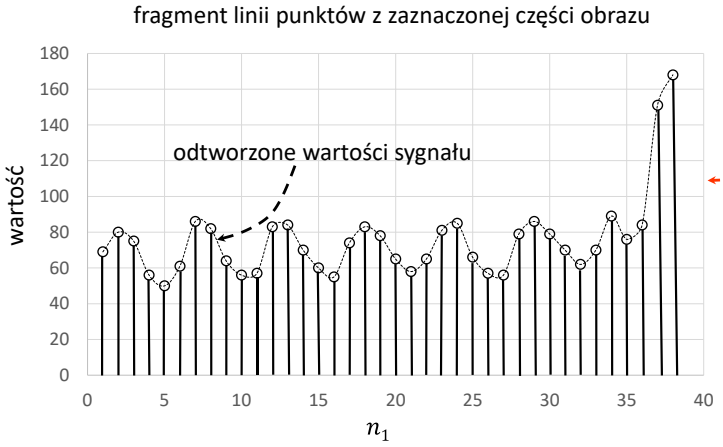
I tak, duże wartości T_1 i T_2 (czyli duże odległości między rejestrowanymi próbkami sceny) odpowiadają niskim częstotliwościom próbkowania obrazu (obrazu sceny). Małe odległości T_1 i T_2 pomiędzy sąsiednimi próbkami to wysoka częstotliwość pobierania próbek z obrazu. Jest intuicyjnie zrozumiałe, że w przypadku dużych odległości pomiędzy próbkami spróbkowanego obrazu, zachowanie w obrazie skokowych zmian wartości sygnału (czyli krawędzi w obrazie, drobnych szczegółów) będzie w ogólności niemożliwe. Chcąc zapamiętać w obrazie dużą ilość drobnych detali, należy pobierać próbki z zachowaniem niewielkiego odstępu pomiędzy próbkami, czyli zastosować wysoką częstotliwością próbkowania (zarówno w pionie, jak i w poziomie). W konsekwencji, zarejestrowany obraz składać się musi z dużej liczby próbek. Jest zatem zrozumiałe, że to treść obrazu będzie determinować dopuszczalne, maksymalne odległości T_1 i T_2 pomiędzy kolejnymi próbkami w obrazie cyfrowym, które pozwolą na prawidłowe (odtworzenie) odwzorowanie treści obrazu. Zastosowanie zbyt niskich częstotliwości próbkowania f_{s1} i f_{s2} da w ogólności obraz nieostry. Nie będzie również możliwe w tym przypadku prawidłowe odwzorowanie fragmentów treści obrazu o szybkozmiennym charakterze (szybko zmieniające się wartości próbek). Przykładowe, negatywne skutki zastosowania zbyt niskiej częstotliwości próbkowania dla obrazu zostały przedstawione na rysunku 1-5.

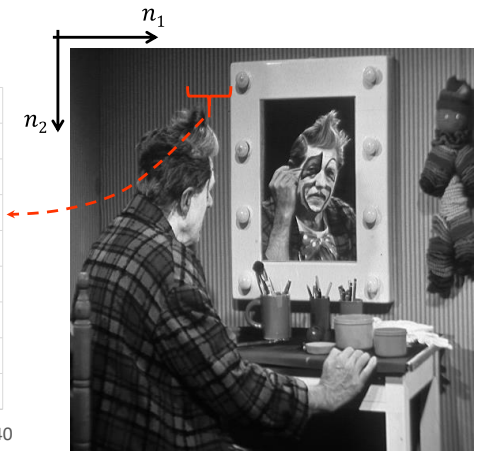


oryginalna częstotliwość próbkowania

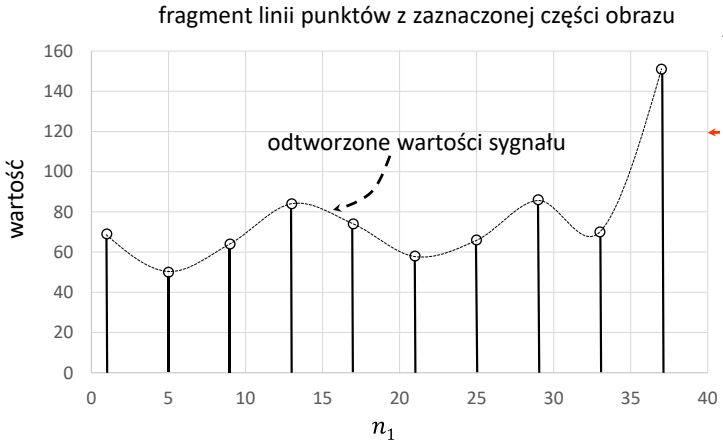


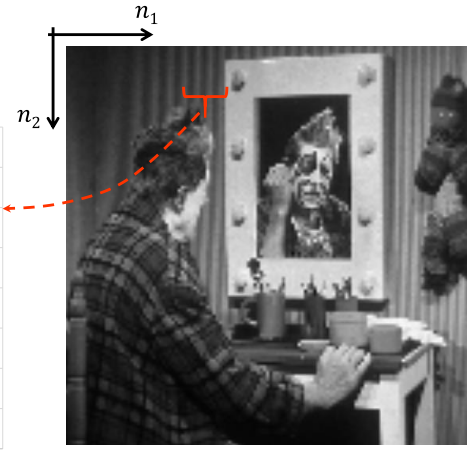
częstotliwość próbkowania zredukowana 4-krotnie w pionie i 4-krotnie w poziomie w stosunku do częstotliwości oryginalnej





oryginalna częstotliwość próbkowania





częstotliwość próbkowania zredukowana 4-krotnie w pionie i 4-krotnie w poziomie w stosunku do częstotliwości oryginalnej

Rysunek 1-5 Wpływ redukcji częstotliwości próbkowania obrazu na jego jakość. Ilustracja faktu błędnego odtworzenia szybkozmiennych fragmentów obrazu, z uwagi na zbyt dużą odległość pomiędzy sąsiednimi rejestrowanymi próbkami obrazu sceny. Przykład opracowany z użyciem własnego oprogramowania.

Z częstotliwościami próbkowania obrazu danej sceny jest związane pojęcie **przestrzennej rozdzielczości obrazu**. Pojęcie to określa ile jest w obrazie wszystkich próbek, czyli, jaka jest liczba próbek w jednej linii próbek obrazu (np. N_1), oraz z jakiej liczby linii próbek składa się obraz (np. N_2). Zgodnie z przyjętymi oznaczeniami przestrzenna rozdzielczość obrazu jest wyrażana jako $N_1 x N_2$. Oczywiście w kontekście tego, o czym powiedziano już wcześniej, im wyższa rozdzielczość obrazu tym lepiej, bo więcej szczegółów możemy w obrazie reprezentować. Należy sobie jednak uświadomić, że uzyskanie obrazów o bardzo wysokich rozdzielczościach przestrzennych wymaga zastosowania nowocześniejszych (czytaj droższych) kamer czy aparatów. Z uwagi na dużą liczbę próbek w takim obrazie, jego zapis, dalsza obróbka czy wreszcie transmisja są również obarczone większymi kosztami. Niemniej jednak, wraz z dokonującym się postępem w zakresie urządzeń rejestrujących obrazy oraz rozwojem technik kompresji obrazów, rozdzielczości przestrzenne obrazów cyfrowych ciągle się zwiększają.

1.3. Kwantyzacja obrazu

1.3.1. Kwantyzacja danych – podstawowe informacje

Istotą kwantyzacji jest zmniejszenie liczby możliwych wartości, jakie może przyjmować sygnał. W przypadku obrazu chodzi o redukcję zbioru możliwych wartości $f(n_1, n_2)$ próbek obrazu (bez kwantowania możliwych wartości próbek byłoby nieskończenie wiele). W tym celu dokonuje się modyfikacji poszczególnych wartości f sygnału w taki sposób, że każdą wartość z zakresu (d_i, d_{i+1}) zastępuje się nową liczbą o jednej, ściśle określonej wartości nazywanej **poziomem reprezentacji** lub **poziomem rekonstrukcji**. Przeprowadzenie kwantyzacji wymaga zatem podzielenia całego zakresu wartości f sygnału na określoną liczbę przedziałów liczbowych (granice tych przedziałów określają tzw. **progi decyzyjne**), a następnie przypisaniu każdemu przedziałowi liczbowemu jednej konkretnej wartości liczbowej (czyli poziomu reprezentacji). W rezultacie otrzymywana jest pewna funkcja „schodkowa”, czyli **charakterystyka kwantyzatora** (patrz rysunek 1-6), która opisuje sposób zamiany wartości wejściowych f na wartości wyjściowe $Q\{f\}$. Ponieważ w omawianym tutaj kwantyzatorze kwantuje się za każdym razem wartość liczbową, czyli skalar, kwantyzator ten jest nazywany **kwantyzatorem skalarnym**.

Na podstawie skwantowanej wartości $Q\{f\}$ nie jesteśmy oczywiście w stanie powiedzieć, jaka dokładnie była wartość f sygnału przed operacją kwantowania. Możemy jedynie powiedzieć, do którego z przedziałów liczbowych należała oryginalna wartość f sygnału. Kwantyzacja jest zatem procesem stratnym, w którym dochodzi do nieodwracalnej utraty części informacji o sygnale, gdyż odtworzona później w dekoderze sygnału wartość $Q\{f\}$ nie będzie w ogólności taka sama jak wartość f przed kwantowaniem. Z kwantowaniem jest więc związany pewien błąd q nazywany **błędem kwantyzacji** lub inaczej **szumem kwantyzacji**, który jest różnicą pomiędzy oryginalną wartością f a wielkością $Q\{f\}$ po kwantyzacji:

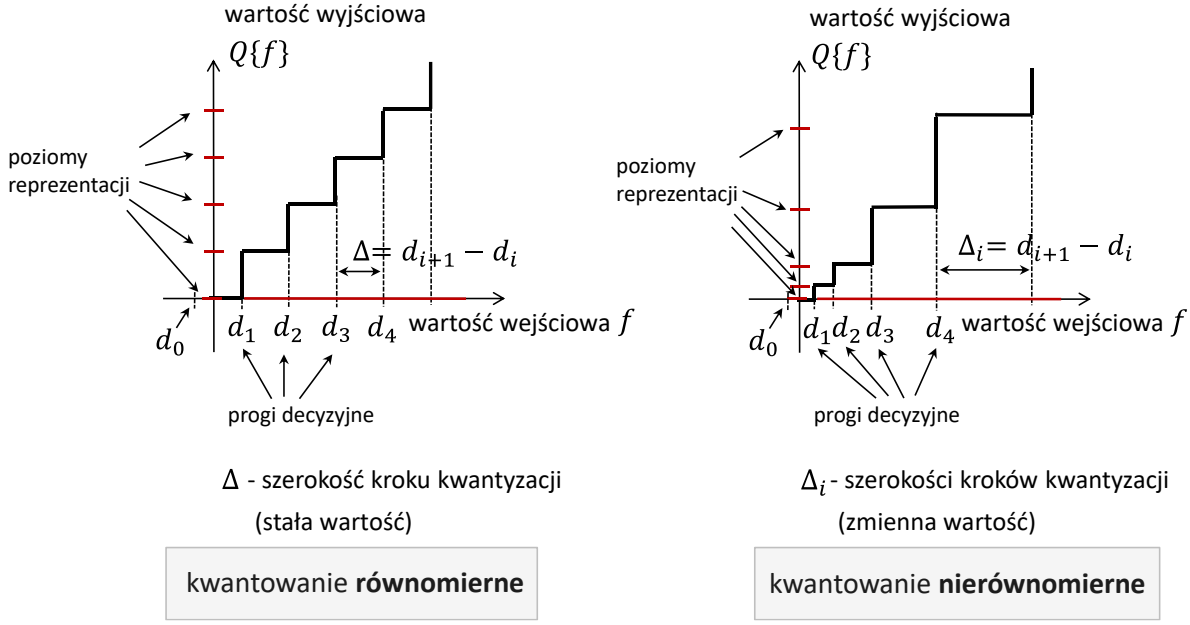
$$\boldsymbol{q = f - Q\{f\}} \tag{1.1}$$

Oczywiście, szum kwantyzacji jest zjawiskiem niepożądanym i dla określonego zbioru wartości wejściowych należy w taki sposób zaprojektować kwantyzator, żeby ten błąd zminimalizować. Na poziom szumu kwantyzacji (czyli wartości błędu q) wpływ mają:

1) Liczba i sposób rozmieszczenia progów decyzyjnych na osi kwantowanych wartości wejściowych, czyli liczba i szerokość przedziałów liczbowych, o których mowa była wcześniej (patrz również rysunek 1-6).

2) Przyjęte dla poszczególnych przedziałów liczbowych wartości poziomów reprezentacji (poziomów rekonstrukcji).

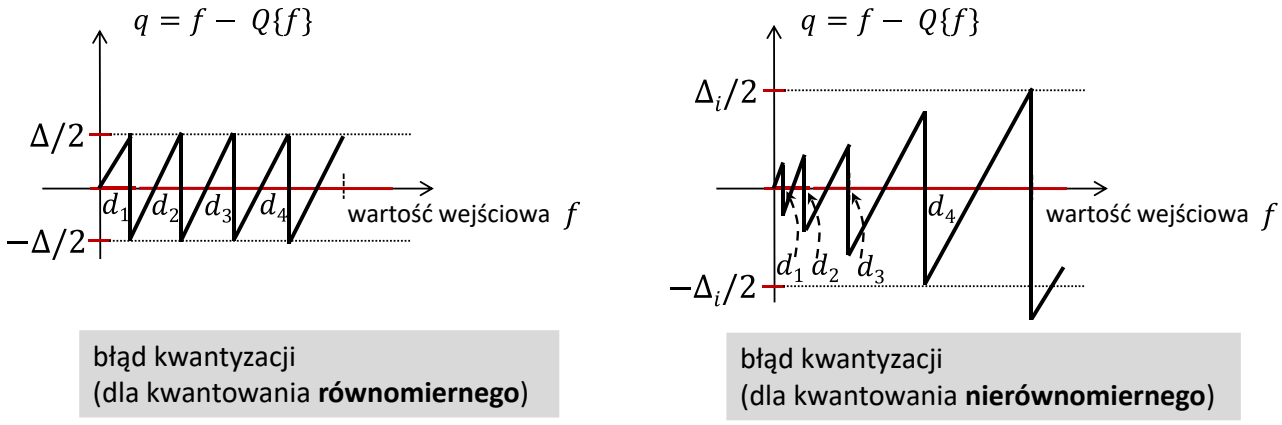
Najprostszym kwantyzatorem jest kwantyzator, w którym przedziały liczbowe mają taką samą szerokość. Szerokość przedziałów liczbowych (oznaczonych na rysunku 1-6 symbolem Δ lub Δ_i) jest nazywana **szerokością kroku kwantyzacji**. Równa szerokość kolejnych przedziałów liczbowych jest wynikiem równomiernego rozłożenia progów decyzyjnych na osi kwantowanych wartości (czyli na osi f), stąd kwantyzator ten jest w literaturze określany mianem **kwantyzatora równomiernego**. W przypadku nierównomiernego rozłożenia progów decyzyjnych, otrzymane przedziały liczbowe mają różną szerokość i mamy wtedy do czynienia z tzw. **kwantowaniem nierównomiernym**. W zależności od tego, ile jest określonych wartości w sygnale (co przekłada się na rozkład prawdopodobieństwa kwantowanych wartości sygnału), oraz jaki rodzaj informacji te wartości reprezentują (np. próbki obrazu, dźwięk, lub inny jeszcze rodzaj sygnału), zaleca się stosowanie konkretnego typu kwantyzatora spośród wymienionych powyżej.



Rysunek 1-6 Przykładowe charakterystyki kwantyzatorów: równomiernego i nierównomiernego.

Ale wróćmy do tematu błędu (szumu) kwantyzacji. Z uwagi na fakt, że w kwantyzatorze każdą liczbę z przedziału (d_i, d_{i+1}) reprezentujemy (z osobna) tą samą wartością liczbową (czyli konkretnym poziomem reprezentacji), to na wartości błędu kwantyzacji, oprócz samego poziomu reprezentacji, bezpośredni wpływ ma szerokość przedziału (d_i, d_{i+1}), czyli szerokość kroku kwantyzacji. Im mniejsza szerokość kroku kwantyzacji, tym mniejsze wartości błędu kwantyzacji, jednak pokrycie całego zakresu wejściowych wartości f wąskimi przedziałami liczbowymi wymaga zastosowania dużej liczby tych przedziałów. Duża liczba przedziałów liczbowych przełoży się z kolei na obszerny zbiór wartości wyjściowych (czyli poziomów reprezentacji), przez co koszt bitowy zapamiętania danej wartości wyjściowej będzie wysoki. Zakładając, że wartości sygnału pochodzące z przedziału (d_i, d_{i+1}), którego szerokość wynosi Δ (lub Δ_i) będą reprezentowane liczbą będącą środkiem przedziału (d_i, d_{i+1}), to błąd kwantyzacji będzie przyjmować wartości z zakresu $(-\Delta/2, +\Delta/2)$ (lub $(-\Delta_i/2, +\Delta_i/2)$), a zerowy błąd otrzymamy tylko dla wartości z środka przedziału (d_i, d_{i+1}). Wykresy błędów kwantyzacji dla przedstawionych na rysunku 1-6

dwóch typów kwantyzatorów (równomiernego i nierównomiernego) zostały przedstawione na rysunku 1-7.



Rysunek 1-7 Wartości błędu kwantyzacji q dla dwóch typów kwantyzatora: równomiernego (rysunek po lewej) oraz nierównomiernego (rysunek po prawej).

Dokładna analiza wykresów błędu kwantyzacji tłumaczy zasadność zastosowania jednego lub drugiego typu kwantyzatora w danym zastosowaniu. Jeśli rozkład statystyczny wartości, które chcemy skwantować jest równomierny (czyli wszystkie wartości pojawiają się z jednakowym prawdopodobieństwem), to optymalnym będzie zastosowanie kwantyzatora równomiernego. W takim przypadku sumaryczny błąd kwantyzacji będzie najmniejszy. W przypadku nierównomiernego rozkładu prawdopodobieństwa kwantowanych wartości lepiej jest zastosować kwantyzator nierównomierny, w którym wartości występujące częściej będzie można skwantować z zastosowaniem mniejszego kroku kwantowania, wprowadzając przez to mniejszy błąd kwantowania. Tyle mówi teoria. W wielu praktycznych zastosowaniach warto jest jednak uwzględnić to, czy z punktu widzenia percepcji kwantowanego sygnału określony zakres wartości tego sygnału nie powinien zostać reprezentowany z większą dokładnością (w porównaniu z innymi zakresami wartości), czyli z „wprowadzeniem” mniejszego błędu kwantowania. Tak właśnie jest w przypadku kwantowania danych, które reprezentują próbki obrazu, o czym będzie się można przekonać czytając następny punkt książki.

1.3.2. Kwantowanie próbek obrazu monochromatycznego

Człowiek dostrzega około 1%-2% zmianę jasności fragmentu obrazu [Wysz82]. Tym samym posiada zdolność rozróżnienia kilkudziesięciu – do stu poziomów jasności tego fragmentu. Jednak, w konkretnym przypadku, zdolność ta może być nieco inna od wskazanej powyżej, ponieważ zależy ona dodatkowo od jasności innych fragmentów obrazu, które otaczają fragment analizowany (mówi się w tym miejscu o zależności od jasności tła).

Dlatego kwantując obraz należy każdej jego próbce przypisać taką liczbę bitów, która uwzględni przytoczone powyżej wnioski. Z jednej strony zależy nam na tym, żeby reprezentacja każdej próbki nie była nadmiarowa (powyżej pewnej liczby bitów na próbkę i tak nie poprawimy już subiektywnej jakości obrazu, a niepotrzebnie zwiększylibyśmy koszt bitowy jego reprezentacji), a z drugiej strony liczba bitów na próbkę nie może być też zbyt mała (bo zbyt mała byłaby liczba reprezentowanych poziomów jasności). W tym drugim przypadku, utracilibyśmy możliwość wiernego przedstawienia fragmentów, w których jasność zmienia się w sposób „płynny” – zamiast

tego zauważylibyśmy skokową (a nie „płynną”) zmianę jasności próbek (patrz przedstawione na rysunku 1-8 porównanie).

reprezentacja **8-bitowa**
(256 poziomów szarości)

reprezentacja **4-bitowa**
(16 poziomów szarości)

Rysunek 1-8 Wpływ liczby bitów reprezentujących próbkę monochromatycznego obrazu na jego jakość. Przykład opracowany z użyciem własnego oprogramowania.

Uwzględniając cechy systemu widzenia człowieka przyjęto w technice obrazu, żeby każdą próbkę obrazu reprezentować na 8 bitach, co daje możliwość przedstawienia $2^8 = 256$ poziomów jasności próbki. W tym przypadku próbki mogą przyjmować wartości z zakresu od 0 do 255. Chociaż w wielu przypadkach taka reprezentacja może być nadmiarowa (patrz przedstawione wcześniej wnioski na temat zdolności postrzegania przez człowieka zmiany jasności fragmentu obrazu), to jednak z punktu widzenia technicznego operowanie (np. odczyt z pamięci i zapis do pamięci) na 8-bitowych próbkach jest wygodne, gdyż 8 bitów tworzy pełen bajt danych. Dlatego reprezentacja taka zdecydowanie przeważa we wszelkich zastosowaniach konsumenckich. Tym samym typowy kwantyzator obrazu monochromatycznego wykorzystuje 256 poziomów reprezentacji dla skwantowanych wartości próbek.

Jednak percepcja (zdolność postrzegania) każdego z 256 poziomów jasności próbek nie jest u człowieka taka sama. Jest ona inna w obrębie próbek o dużej jasności (określanych dalej jako próbki „jasne”) i inna w przypadku próbek, których jasność jest niewielka (próbki „ciemne”). Zgodnie z obserwacjami Webera i Fechnera (wnioski z tych obserwacji są w literaturze znane jako **prawo Webera-Fechnera**) człowiek wykazuje większą wrażliwość na zmianę jasności próbek „ciemnych”, a mniejszą na zmianę jasności próbek „jasnych”. Tym samym wprowadzenie w obrębie „ciemnych” próbek określonego zniekształcenia (zaszumienia) będzie łatwiej zauważalne niż gdyby to samo zniekształcenie pojawiło się w obrębie próbek „jasnych”. Z tego właśnie powodu próbki „ciemne” powinno się kodować z wyższą dokładnością w porównaniu do precyzji kodowania próbek „jasnych”. Dlatego uzasadnionym jest **zastosowanie dla obrazu nierównomiernego kwantyzatora** próbek, w którym próbki o małej wartości są kwantowane z zastosowaniem relatywnie małych kroków kwantyzacji (czyli w efekcie mniejszy szum

kwantowania), a próbki o wartości dużej z użyciem kroków kwantyzacji odpowiednio większych (czyli większe wartości szumu kwantowania).

1.4. Reprezentacje cyfrowego obrazu kolorowego

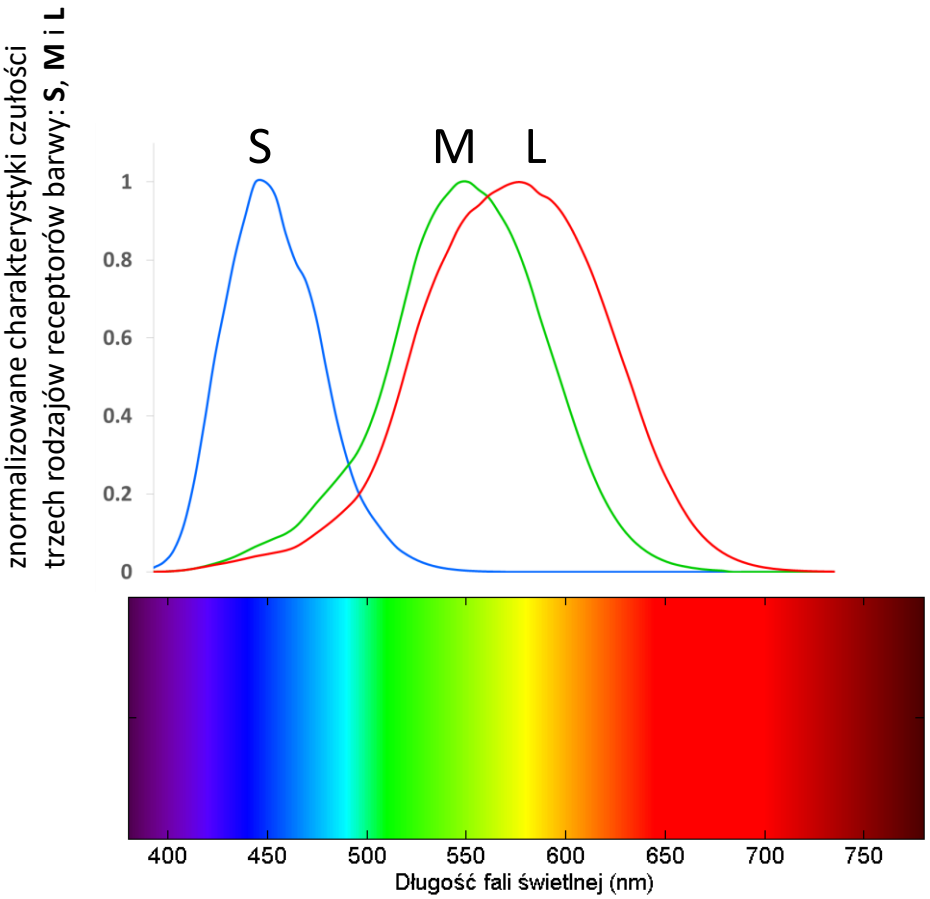
1.4.1. Krótkie wprowadzenie

Zapewne każdy pamięta przeprowadzane na lekcjach fizyki doświadczenie, podczas którego za pomocą pryzmatu dokonywano rozszczepienia światła białego na wiele promieni świetlnych, z których każdy posiadał inną „barwę”. Przytoczone zjawisko zaobserwować można także na niebie w postaci tęczy, kiedy to dochodzi do rozszczepienia białego światła na zawieszonych w powietrzu kropelkach wody. Doświadczenia te dowodzą faktu, iż **białe światło jest mieszaniną „świateł” o różnych barwach**, z których każde jest falą (promieniem) o ściśle określonej długości fali[3] (co przekłada się na częstotliwość tej fali). To właśnie długość fali promienia świetlnego decyduje o tym, jakie wrażenie barwne promień ten wywoła w ludzkim oku. Dla wybranego zestawu barw związek pomiędzy postrzeganym przez człowieka „kolorem” światła a długością fali promienia świetlnego można odczytać z rysunku 1-9.

Elementami w ludzkim oku, które bezpośrednio odpowiadają za postrzeganie kolorów (barwy światła) są tzw. receptory barwy, które w biologii są nazywane czopkami. Badania anatomii oka wskazują, iż u człowieka daje się wyróżnić **trzy rodzaje receptorów barwy**, które są szczególnie czułe na światło w kolorze **niebieskim**, **zielonym** i **czerwonym**. Analizując przedstawione na rysunku 1-9 charakterystyki czułości trzech rodzajów czopków (tzw. czopki S, M i L) [4] dobrze widać, że przyjmują one maksimum w zakresach długości fali świetlnej, które są związane z wymienionymi barwami.

[3] Możliwość „wyłuskania” ze światła białego promieni świetlnych o poszczególnych barwach wykazał Newton w latach 1665-1666.

[4] Przyjęte w literaturze oznaczenia czopków (**S**, **M** i **L**) pochodzą od pierwszych liter angielskich terminów wyrażających zakresy długości fali świetlnej (fale „krótkie” – **S**hort, fale „średnie” – **M**edium, fale „długie” – **L**ong), w których dany rodzaj czopka wykazuje największą wrażliwość (czułość).



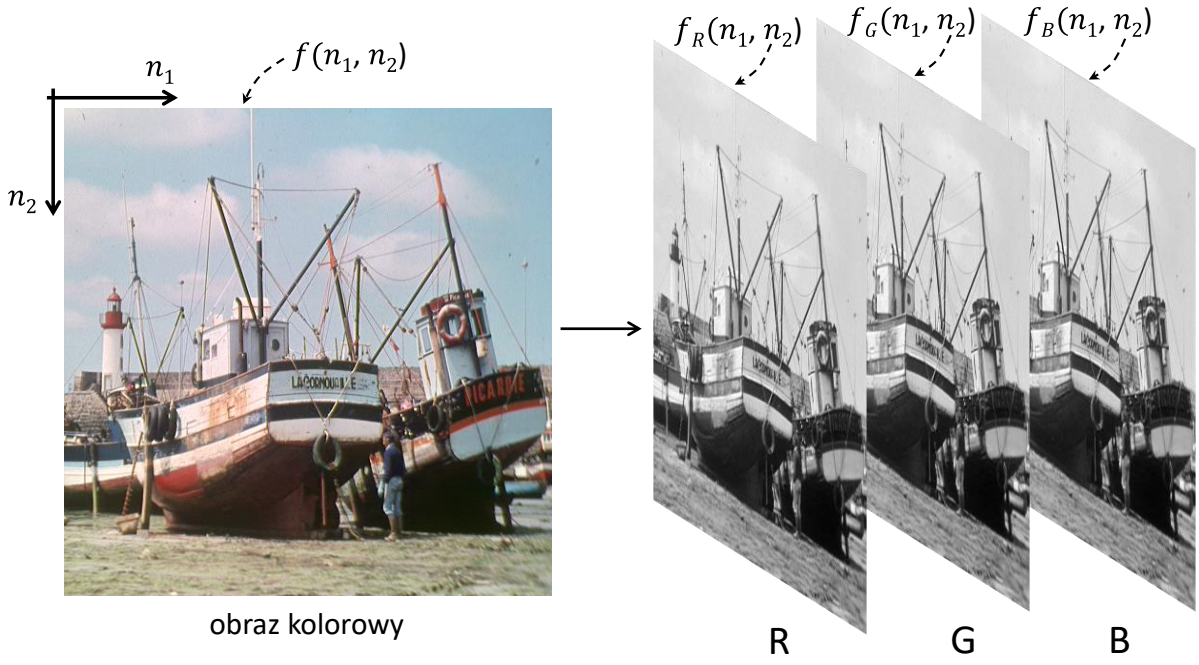
Rysunek 1-9 Znormalizowane charakterystyki czułości trzech rodzajów receptorów barwy (czopków), jakie występują w ludzkim oku. Wykres został opracowany na podstawie danych zamieszczonych w rozprawie doktorskiej Yuta Asano [Asano15] (zgodnie z modelem widza-obserwatora, jaki został zaproponowany w normie CIE 170-1:2006).

Stąd, bardzo często mówi się o występujących w ludzkim oku receptorach barwy **niebieskiej**, **zielonej** i **czerwonej**. W zależności od tego, z jaką intensywnością światło pobudza każdy z trzech typów receptorów wywołuje to u człowieka wrażenie określonej barwy. Barwa ta jest więc **sumą odpowiedzi receptorów barwy danego typu na pobudzenie światłem**. Wrażenie barwne, jako odpowiedź na pobudzenie trzech typów receptorów w oku (tak jak ma to miejsce u człowieka), jest w literaturze nazywane **trójchromatycznym widzeniem barwnym**. Taki sposób widzenia barw (czyli w oparciu o określone trzy barwy podstawowe) jest podstawą przyjętej w technice metody reprezentacji obrazu kolorowego (także cyfrowego).

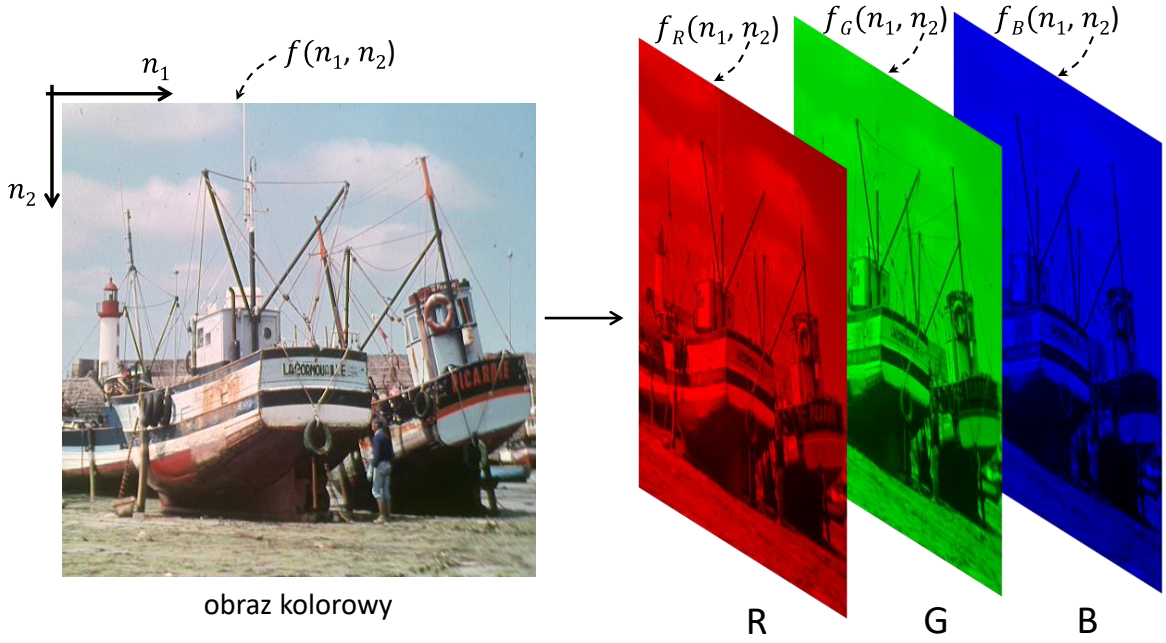
1.4.2. Obraz kolorowy w przestrzeni RGB

Z przedstawionych powyżej powodów światło kolorowe jest w technice obrazu uzyskiwane poprzez „zmieszanie” ze sobą, w odpowiednich proporcjach, nie dużej liczby składowych barwnych, ale **trzech barw podstawowych**, którym odpowiadają światła o ściśle określonej barwie. Z uwagi na obecne w ludzkim oku rodzaje czopków (patrz poprzedni punkt) tymi trzema barwami podstawowymi są barwy: **czerwona** (**R**ed), **zielona** (**G**reen) i **niebieska** (**B**lue). Taki sposób otrzymywania kolorowego obrazu, chociaż bardzo uproszczony, jest od dawna wykorzystywany w praktyce i jest określany jako reprezentacja obrazu w **przestrzeni RGB**.

Zgodnie z tą reprezentacją, z każdą próbką kolorowego obrazu związane są trzy liczby (po jednej dla składowej R, G i B), które określają, jakie są proporcje (i wartości absolutne) „świateł” czerwonego, zielonego i niebieskiego w wynikowym świetle kolorowym. Ponieważ taką informację należy zapisać dla każdej próbki kolorowego obrazu, to można powiedzieć, że obraz kolorowy składa się z trzech monochromatycznych obrazów, które określają dla każdej kolorowej próbki proporcje (i wartości absolutne) składowych R, G i B (patrz rysunek 1-10). Każdy z tych trzech monochromatycznych obrazów informuje więc o „intensywności” świateł czerwonego (R), zielonego (G) i niebieskiego (B) w kolorowym obrazie, co zostało dodatkowo zobrazowano na rysunku 1-11.



Rysunek 1-10 Obraz kolorowy w przestrzeni kolorów (reprezentacji) RGB. Wartości poszczególnych składowych (R, G i B) mogą być reprezentowane za pomocą trzech monochromatycznych obrazów.

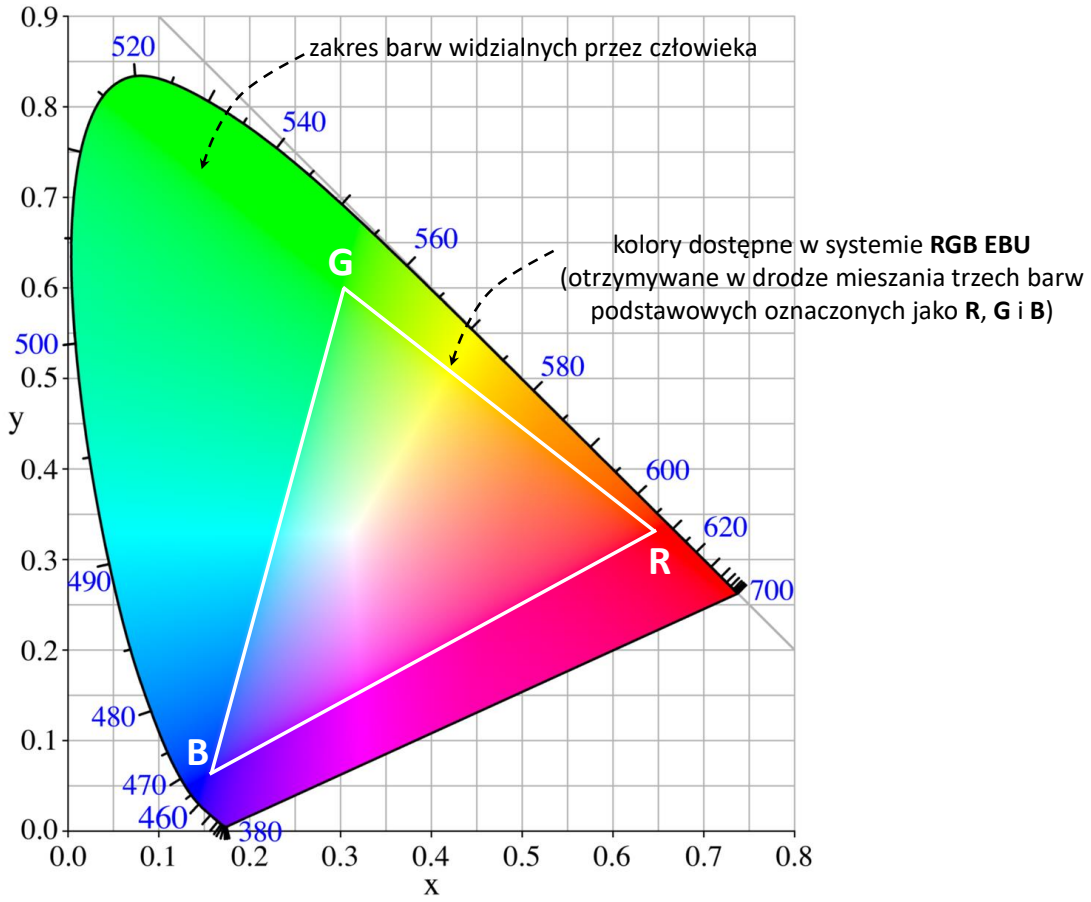


Rysunek 1-11 Obraz kolorowy w przestrzeni RGB. Poszczególne obrazy składowych informują o „intensywności” światła czerwonego (R), zielonego (G) i niebieskiego (B) w kolorowym obrazie.

W przypadku kolorowego obrazu, który jest na przykład przesyłany w telewizji, pojedyncza próbka każdej składowej (R, G i B) jest zwykle reprezentowana na 8 bitach, co daje w sumie 24 bity na jedną próbkę obrazu kolorowego. Taki sposób reprezentacji kolorowego obrazu pozwala uzyskać relatywnie wysoką jego jakość, i w związku z tym przeważa w zastosowaniach konsumenckich. Stosowanie większej dokładności reprezentacji próbki (np. 10, 12, czy 16 bitów na próbkę jednej składowej) jest uzasadnione w zastosowaniach profesjonalnych (np. w studiu telewizyjnym), gdzie przed ostateczną transmisją obrazu do użytkownika przechodzi on zwykle wiele etapów przetwarzania (filtracji), z czym wiążą się określone błędy zaokrągleń. W takich sytuacjach, większa dokładność reprezentacji próbki pozwala te błędy istotnie zmniejszyć.

1.4.3. Obraz kolorowy w przestrzeni RGB – ograniczenia reprezentacji barw

Fakt, że w przestrzeni (modelu) RGB kolorowy obraz jest tworzony w oparciu o tylko trzy barwy podstawowe (a nie większą ich liczbę) narzuca istotne ograniczenia na liczbę możliwych kolorów, jakie daje się w ten sposób odwzorować. Mówiąc najkrócej, w reprezentacji RGB możliwe jest odwzorowanie tylko części pełnego zakresu barw widzialnych, co zostało przedstawione na rysunku 1-12.



Rysunek 1-12 Figura przedstawiająca zakres barw widzialnych przez człowieka [5]. Białym trójkątem zaznaczono podzbiór barw (tzw. **gamę barw**, czyli **gamut**), które daje się otrzymać w drodze ważonego sumowania trzech barw podstawowych: R, G i B (dotyczy opracowanego przez Europejską Unię Nadawców (EBU – European Broadcasting Union) systemu RGB EBU). Przygotowano na podstawie opracowania użytkownika Internetu o pseudonimie PAR.

Na tym rysunku, przypominająca podkowę figura przedstawia zakres wszystkich widzialnych barw, które jest w stanie odróżnić tzw. standardowy obserwator[6]. Jak widać, w przypadku opracowanego w Europie systemu RGB EBU, możliwe do odwzorowania barwy mieszczą się w obrębie białego trójkąta (**co stanowi tylko 35% pełnego zakresu barw widzialnych**), przez co niemożliwe staje się odwzorowanie dużej części zakresu widzialnych kolorów. Ograniczenie to, dla rozważanego systemu RGB, w sposób szczególny dotyczy wielu odcieni kolorów zielonego, purpurowego i błękitnego. Tych kolorów nie wyświetli więc żaden monitor czy projektor, który działa w oparciu o zgodne z systemem RGB EBU trzy barwy podstawowe. Sformułowany tutaj wniosek pozostaje prawdziwy także dla innych odmian systemu RGB, których bardzo krótki przegląd został zamieszczony w punkcie 1.4.4.

Pomimo aż tak istotnego zawężenia zakresu możliwych do odwzorowania barw, oferowana przez systemy RGB gama barw spełnia oczekiwania zdecydowanej większości zwykłych

[5] Jest to figura barw widzialnych, sporządzona w układzie współrzędnych trójchromatycznych xy. Matematyczny opis współrzędnych xy znaleźć można w pracy [Doma10].
[6] Pojęcie ‘standardowego obserwatora’ pojawiło się w opracowanym przez Międzynarodową Komisję Oświetleniową systemie CIE 1931 RGB (rok 1931). Określa ono (uśrednione) możliwości percepcji barw u osób, które prawidłowo rozróżniają kolory.

użytkowników. Z pewnością większość z nich nie jest nawet świadoma omawianych tutaj ograniczeń przestrzeni RGB.

1.4.4. Przestrzeń RGB – odmiany i zastosowanie

Chociaż właściwa idea tworzenia kolorów na podstawie zdefiniowanych wcześniej trzech barw podstawowych jest taka jak przedstawiono to powyżej, to na przestrzeni lat opracowanych zostało wiele różnych odmian systemu RGB. Poszczególne odmiany różnią się definicją barw podstawowych (długość fali światła dla barwy czerwonej, zielonej i niebieskiej) i zostały opracowane dla ściśle określonych zastosowań. I tak przykładowo, opracowane w USA systemy RGB (np. **RGB NTSC**) różniły się od tych stosowanych w Europie (np. **RGB EBU**). Stosowane w telewizji cyfrowej systemy barw też nie są identyczne z tymi, jakie stosowano dla analogowego sygnału wizyjnego. Obecnie bardzo szerokie zastosowanie ma system **RGB** zdefiniowany w zaleceniu Międzynarodowej Unii Telekomunikacyjnej **ITU-T BT.709**, który jest tożsamy z tzw. standardowym systemem RGB – **sRGB** (który jest wynikiem współpracy firm Hewlett-Packard i Microsoft). Ponieważ gama barw systemu sRGB odpowiada z dużą dokładnością zakresowi kolorów, który jest możliwy do wyświetlenia na wielu monitorach komputerowych, stanowi on aktualnie podstawę reprezentacji cyfrowych obrazów, które są wymieniane w sieci Internet.

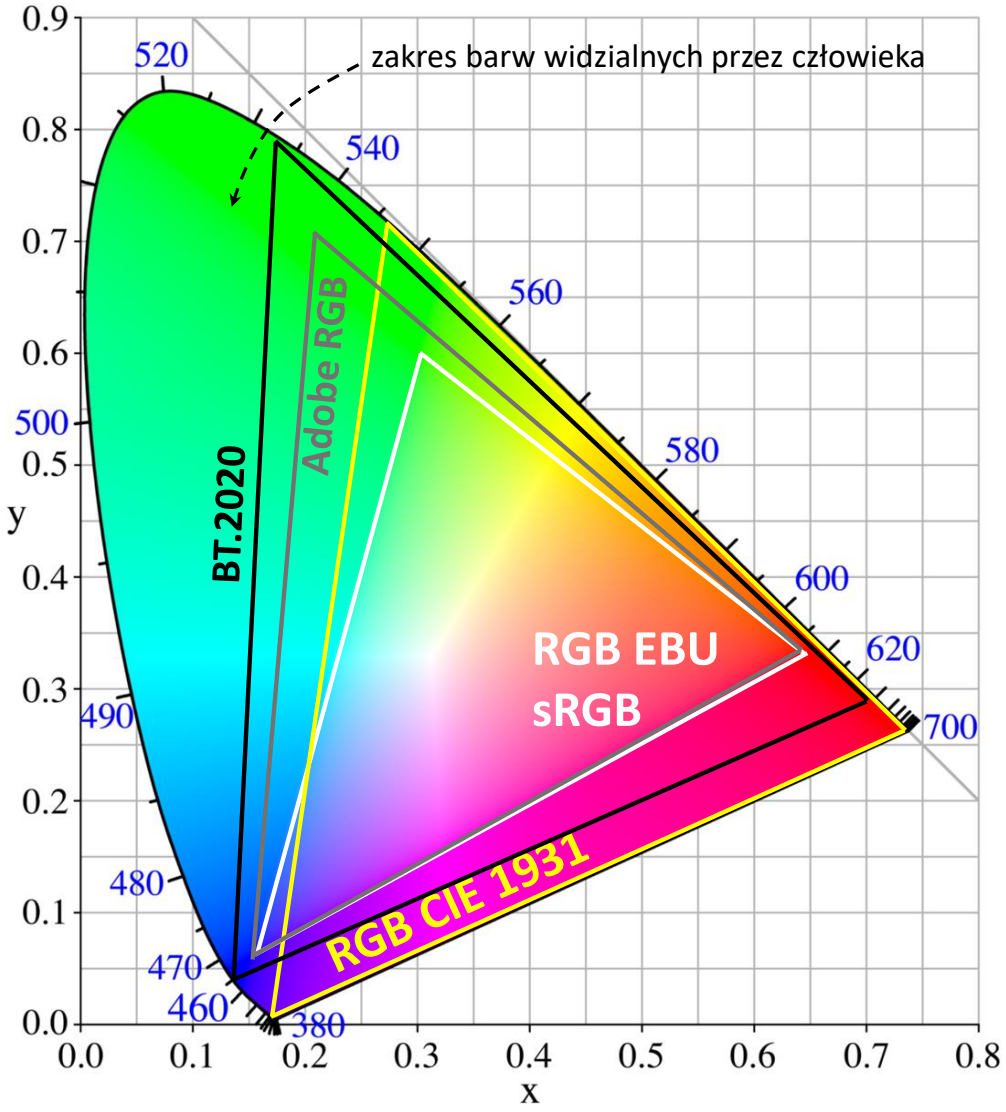
Spośród dostępnych odmian systemu RGB na przywołanie zasługuje system **Adobe RGB**. System Adobe RGB[7] ma szczególne znaczenie dla cyfrowej fotografii, gdyż jego zakres barw, który jest znacznie szerszy w porównaniu z wieloma innymi systemami RGB, w dużo większym stopniu odpowiada gamie barw systemu **CMYK** (ang. Cyan, Magenta, Yellow, Key Colour), który jest stosowany w urządzeniach drukujących. Jako ciekawostkę można tutaj dopowiedzieć, że przestrzeń Adobe RGB zdolna jest reprezentować przeszło 50% pełnego zakresu barw widzialnych, podczas gdy w przypadku systemów RGB EBU i sRGB jest to tylko 35%. W sposób szczególny przekłada się to w systemie Adobe RGB na możliwość odwzorowania większej liczby odcieni kolorów zielonego, niebieskiego oraz kolorów z pogranicza zielony-niebieski.

Mówiąc o różnych systemach RGB nie sposób pominąć systemu **CIE 1931 RGB**, który został opracowany przez Międzynarodową Komisję Oświetleniową już w roku 1931. System ten odwzorowuje blisko 60% całego zakresu barw widzialnych i jest systemem szczególnym, gdyż jego podstawą są trzy składowe barwne, które dokładnie są **światłami monochromatycznymi**, czyli takimi, które mają jedną, ściśle określoną długość fali λ. W przypadku tego systemu są to składowe: czerwona (λ=700 nm), zielona (λ=546,1 nm), niebieska (λ=435,8 nm). Należy zwrócić uwagę, że w przypadku pozostałych systemów RGB wymienionych wcześniej w tym punkcie, barwy podstawowe są **wielobarwnymi światłami**, gdyż stanowią pewną „mieszankę” świateł o różnej długości fali. Chociaż system CIE 1931 RGB nie jest wykorzystywany w praktyce (np. w urządzeniach wyświetlających obrazy stosuje się inne systemy RGB), to stanowi on jednak punkt odniesienia dla oceny innych systemów RGB.

Dokonujący się w technice telewizyjnej postęp otwiera coraz to większe możliwości reprezentacji i wyświetlania kolorowego obrazu o bardzo szerokim zakresie barw. Dobrym przykładem jest tutaj zdefiniowana w normie **BT.2020** przestrzeń barw, która została opracowana w roku 2012 na potrzeby telewizji bardzo wysokiej jakości obrazu (**UHDTV** – Ultra High Definition Television). W ramach tej przestrzeni istnieje możliwość odwzorowania aż 76% wszystkich kolorów, które może zobaczyć człowiek, co ma zagwarantować nieosiągalny dla wcześniejszych systemów RGB realizm wyświetlanego obrazu.

[7] System ten został opracowany w 1998 roku przez firmę Adobe Systems.

Porównanie możliwej do odwzorowania gamy barw dla kilku szczególnie popularnych systemów RGB zostało przedstawione na rysunku 1-13.



Rysunek 1-13 Możliwa do uzyskania w wybranych systemach RGB gama barw. Porównanie przestrzeni barw systemów: RGB CIE 1931, RGB EBU, sRGB, Adobe RGB. Uwaga, chociaż przestrzenie RGB EBU i sRGB nie są identyczne, to zakresy barw tych systemów są prawie takie same. Przygotowano na podstawie opracowania użytkownika Internetu o pseudonimie PAR.

Przestrzenie RGB znajdują zastosowanie we wszelkiego rodzaju urządzeniach wyświetlających obrazy (monitory, telewizory czy projektory), których zasada działania sprowadza się do emisji świateł o zdefiniowanych wcześniej barwach podstawowych. W oparciu o modele RGB działają również urządzenia rejestrujące i analizujące obrazy, jak np. cyfrowe aparaty, kamery, czy skanery obrazu.

1.4.5. Obraz kolorowy w przestrzeni YC_BC_R

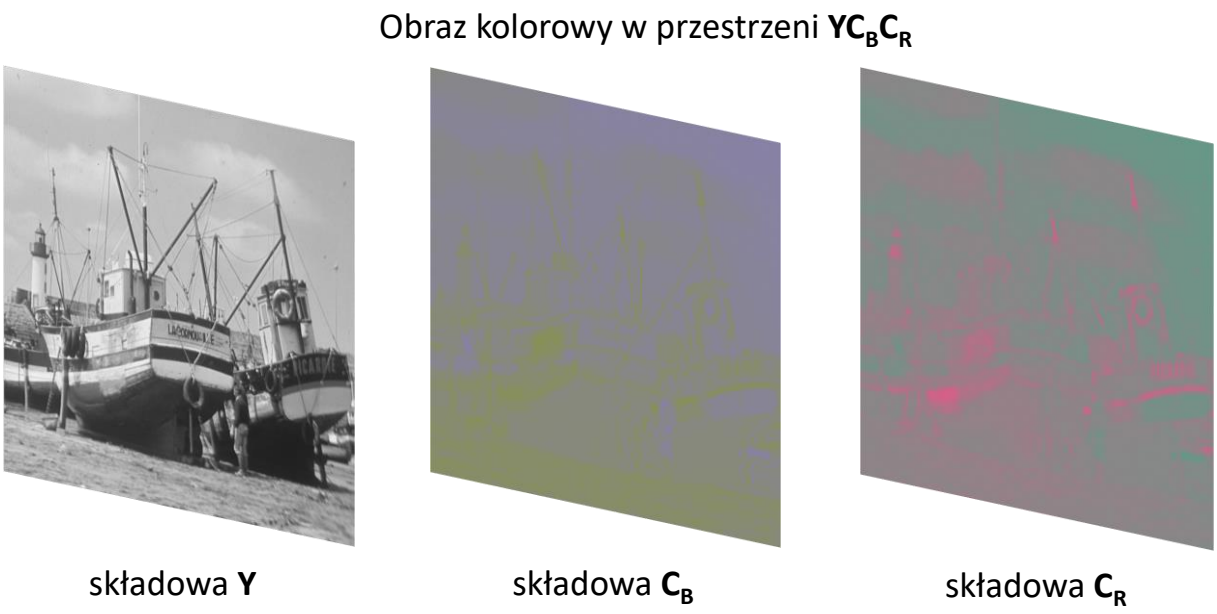
Analizując treść trzech obrazów (obrazy R, G i B na rysunkach 1-10 i 1-11), które opisują poszczególne składowe barwne w reprezentacji RGB, nietrudno jest zauważyć, że obrazy te są do siebie bardzo podobne. Mówi się, że pomiędzy poszczególnymi składowymi przestrzeni RGB istnieje silna korelacja. W tej sytuacji, niezależne kodowanie (celem np. przesłania obrazu kolorowego do dekodera) każdej ze składowych R, G, B obrazu kolorowego byłoby bardzo nieefektywne, gdyż w efekcie trzy razy kodowalibyśmy dane, które w niewielkim tylko stopniu się od siebie różnią. Z tego właśnie powodu transmisja obrazu czy jego zapis na dysku są poprzedzone odpowiednią modyfikacją składowych RGB, celem przeprowadzenia wstępnej dekorelacji trzech obrazów składowych. Chodzi o uzyskanie takiej nowej reprezentacji kolorowego obrazu, w której poszczególne składowe obrazu nie będą wykazywać aż tak dużego podobieństwa.

Powszechną praktyką realizacji tego celu jest zastosowanie dla składowych R, G, B relatywnie prostego matematycznego przekształcenia, którego wynikiem są nowe składowe obrazu, określane jako **Y**, $\mathbf{C_B}$, $\mathbf{C_R}$. Wspomniane przekształcenie można sprowadzić do wykonania następujących kroków obliczeniowych:

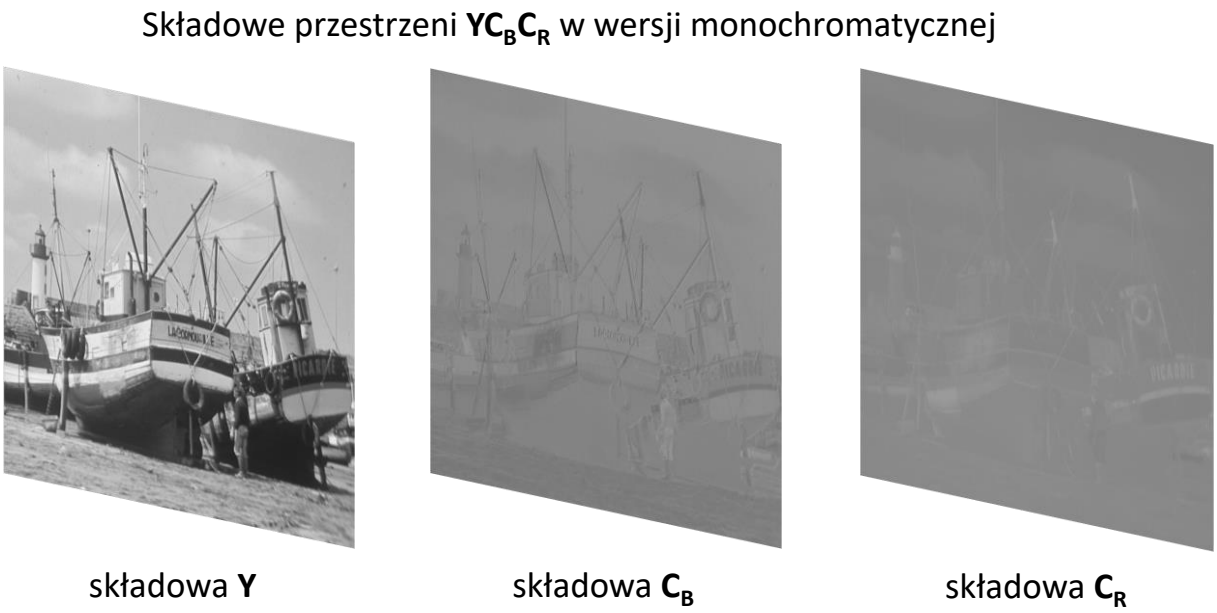
$$\begin{aligned} Y &= 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B \\ C_B &= 128 - (0{,}168736 \cdot R) - (0{,}331264 \cdot G) + (0{,}5 \cdot B) \\ C_R &= 128 + (0{,}5 \cdot R) - (0{,}418688 \cdot G) - (0{,}081312 \cdot B) \end{aligned} \qquad (1.2)\ [8]$$

Otrzymana w ten sposób nowa przestrzeń barw, $\mathbf{YC_BC_R}$, jest szeroko stosowana w systemach transmisji i przechowywania obrazu, i z tego właśnie powodu jest określana mianem **przestrzeni współrzędnych transmisyjnych**. W tej przestrzeni kolorowy obraz jest reprezentowany przy pomocy składowej **luminancji** Y, która określa stopień jasności poszczególnych części obrazu, oraz dwóch różnicowych składowych **chrominancji** (koloru) C_B i C_R, które przenoszą informację o występujących w obrazie kolorach. Postać obrazów poszczególnych składowych została przedstawiona na rysunku 1-14 dla przykładowego obrazu testowego. Rysunek 1-15 pokazuje, jak wyglądają poszczególne obrazy składowych Y, C_B, C_R w ujęciu monochromatycznym.

[8] Składowe przestrzeni YC_BC_R są wyznaczane w oparciu o składowe R, G i B, które zostały wcześniej poddane tzw. korekcji gamma [Doma10].



Rysunek 1-14 Obraz kolorowy w przestrzeni YC_BC_R. Kolejne obrazy przedstawiają składowe: luminancji Y oraz dwie różnicowe składowe chrominancji C_B i C_R.



Rysunek 1-15 Składowe przestrzeni YC_BC_R przedstawione jako obrazy monochromatyczne.

Podobnie jak miało to miejsce w przypadku przestrzeni RGB, znanych jest wiele odmian przestrzeni barw, w których faktycznie wyróżnia się składową luminancji Y i dwie składowe chrominancji (oznaczone np. jako C_B i C_R). Przykładowo, opisana równaniem 1.2 przestrzeń barw jest wykorzystywana w systemie wymiany plików JPEG (JPEG jest standardem kompresji statycznego obrazu). Gdybyśmy porównali jednak tę przestrzeń z innym systemem, który jest stosowany w telewizji cyfrowej wysokiej jakości (High Definision Television – HDTV) [ITU-R BT.709], czy z systemem, który miał zastosowanie w USA [SMPTE 240M], to zobaczylibyśmy relatywnie niewielkie, ale jednak różnice w matematycznym opisie tych przestrzeni. W każdym

jednak przypadku, największy udział w wyznaczeniu składowej luminancji Y przypada składowej zielonej G przestrzeni RGB (patrz wartości współczynników w równaniu 1.2), co wynika bezpośrednio ze szczególnej wrażliwości człowieka na światło o tym właśnie kolorze.

1.4.6. Kompresja kolorowego obrazu – dlaczego właśnie w przestrzeni YC_BC_R?

Zaprezentowane w poprzednim punkcie obrazy składowych Y, C_B, C_R rzeczywiście znacząco się od siebie różnią. Nie wykazują one dużego podobieństwa, co jest wręcz regułą w przypadku obrazów składowych R, G i B. Dlatego, realizowane w ramach przestrzeni YC_BC_R kodowanie obrazu daje nam gwarancję tego, że dany rodzaj informacji (treści) obrazu zostanie zakodowany jeden raz. Można inaczej powiedzieć, że w obrazach składowych Y, C_B, C_R nie ma podobnej do siebie informacji, której kodowanie byłoby powtórzone na etapie kompresji obrazu kolejnej składowej. Jednak nie jest to jedyny powód wykorzystania przestrzeni współrzędnych transmisyjnych w zadaniach kompresji obrazu.

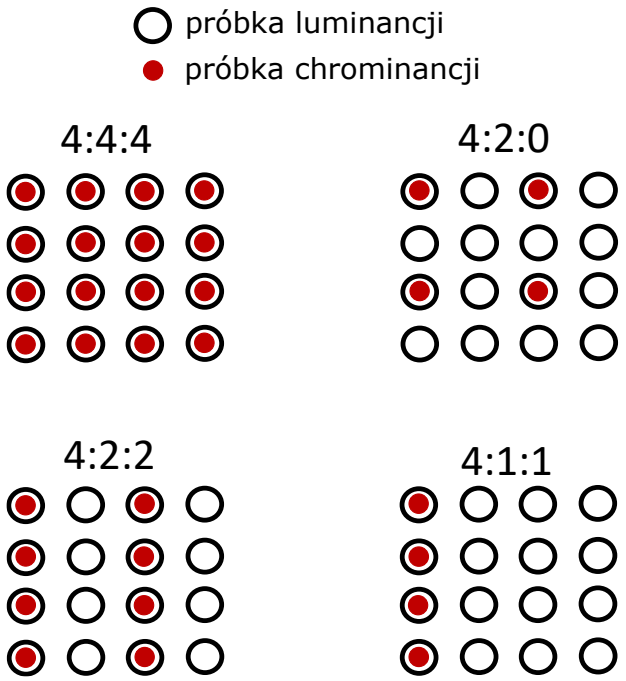
Kolejny powód wynika wprost z różnej istotności informacji o poziomie jasności fragmentów obrazu (zapisanej w składowej Y) oraz informacji o kolorze tych fragmentów (która jest zapisana w składowych C_B i C_R), z punktu widzenia percepcji obrazu przez człowieka. Z tej perspektywy największe znaczenie ma informacja o poziomie jasności fragmentów obrazu, czyli składowa Y, podczas gdy ranga informacji o kolorze jest znacznie mniejsza. Zgodnie z tą obserwacją zachodzi uzasadnienie dokładniejszego reprezentowania składowej Y obrazu, a składowe C_B i C_R można kodować z dokładnością mniejszą. Wspomniane zróżnicowanie dokładności reprezentowania składowych przestrzeni YC_BC_R jest bardzo często stosowaną praktyką w zadaniach kompresji obrazu, ponieważ daje wymierne korzyści w postaci zmniejszonego strumienia bitowego zakodowanych danych, przy zachowaniu ciągle wysokiej jakości zakodowanych obrazów.

W kontekście przyjętego tytułu punktu nie bez znaczenia jest również sam fakt prostoty matematycznych formuł, które definiują przestrzeń YC_BC_R (patrz równania 1.2). Formuły tej przestrzeni są na sztywno określone, czyli ich definicja nie zależy od treści obrazu. W takiej sytuacji nie ma konieczności każdorazowego wyznaczania definicji przestrzeni (przekształcenia) dla obrazów i jej przesyłania do dekodera.

1.4.7. Przestrzeń YC_BC_R i schematy próbkowania chrominancji

Istotną konkluzją poprzedniego punktu jest możliwość bardziej zgrubnego kodowania informacji o kolorze w obrazie, przy zapewnieniu ciągle wysokiej jakości obrazu zakodowanego. Spośród wielu sposobności realizacji tego celu, koncepcyjnie najprostszym wydaje się być pobieranie próbek składowych koloru C_B i C_R z częstotliwością próbkowania mniejszą w porównaniu ze sposobem próbkowania składowej luminancji Y. Dokonujemy w ten sposób bezpośredniej redukcji ilości danych, które informują o zawartych w obrazie barwach.

W praktyce kompresji obrazu najczęściej stosowanym **sposobem (schematem) próbkowania składowych koloru** jest tzw. schemat **4:2:0**. Ilustracja tego schematu została przedstawiona na rysunku 1-16.



Rysunek 1-16 Umiejscowienie próbek luminancji i chrominancji w wybranych schematach próbkowania chrominancji.

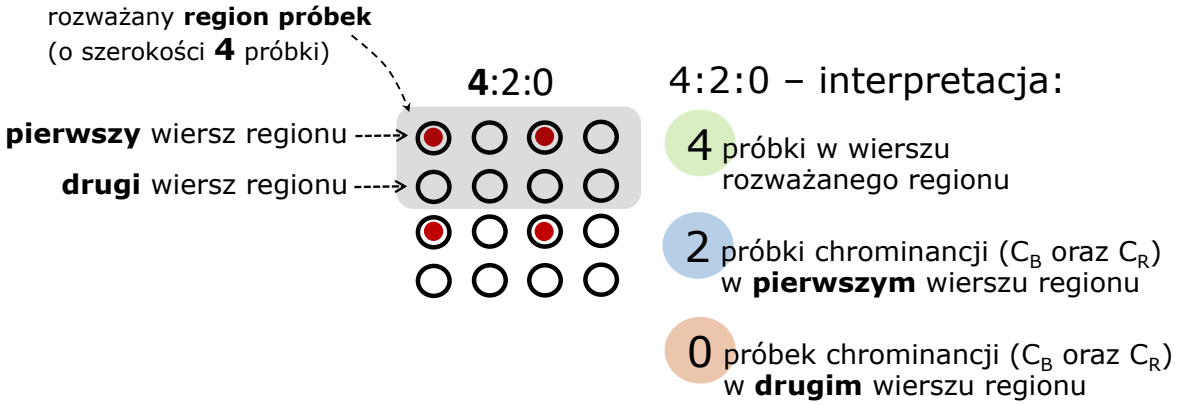
Zgodnie z nim z oryginalnych obrazów składowych C_B i C_R (których przestrzenna rozdzielczość jest taka sama jak rozdzielczość składowej Y) usuwa się co drugą próbkę w pionie oraz co drugą próbkę w poziomie. Wartości pozostałych, nieusuniętych próbek składowych koloru są zwykle odpowiednio zmieniane w drodze dolnoprzepustowej filtracji próbek. W efekcie tych operacji otrzymuje się zmienione obrazy składowych C_B i C_R, których przestrzenna rozdzielczość (rozdzielczość każdej składowej koloru z osobna) jest dwukrotnie mniejsza w pionie i dwukrotnie mniejsza w poziomie, w stosunku do rozdzielczości składowej Y. Tak więc każda składowa koloru liczy w schemacie 4:2:0 cztery razy mniej próbek w porównaniu z liczbą próbek składowej Y. Pomimo aż tak silnego zredukowania ilości informacji reprezentującej kolor, jakość końcowego obrazu kolorowego nie budzi zastrzeżeń statystycznego odbiorcy, a wynikające z uproszczonej reprezentacji barwy oszczędności bitowe będą bardzo wyraźne. O braku zasadniczego wpływu próbkowania 4:2:0 na jakość finalnego obrazu można się łatwo przekonać porównując jakość dwóch obrazów (patrz rysunek 1-17), z których jeden został odtworzony w oparciu o składową Y i pełną (niezredukowaną) rozdzielczość składowych C_B i C_R, a drugi na podstawie składowej Y i składowych C_B i C_R o rozdzielczości wcześniej zredukowanej.

Rysunek 1-17 Schemat próbkowania 4:2:0 i uzyskiwana jakość odtworzonego obrazu.

Z tego właśnie powodu próbkowanie **4:2:0** jest powszechną praktyką reprezentacji barwy w obrazach kolorowych, które są przesyłane w telewizji, czy w sieci Internet. Tak jest w przypadku zastosowań konsumenckich, czyli przewidzianych dla masowego odbiorcy. W zastosowaniach profesjonalnych (np. studio telewizyjne, postprodukcja filmowa) zachodzi potrzeba wierniejszego niż ma to miejsce w schemacie 4:2:0 reprezentowania barwy, i w takich przypadkach spotkać się można ze schematem próbkowania **4:2:2**, czy nawet **4:4:4**. Sposoby próbkowania próbek koloru dla wymienionych schematów zostały również przedstawiony na rysunku 1-16.

1.4.8. Schematy próbkowania chrominancji – jaki jest sens przyjętych oznaczeń liczbowych?

Chociaż stosowanie trzech liczb (np. **4:2:0**) do opisu sposobu próbkowania informacji o kolorze jest w technice obrazu praktyką powszechną, to jednak dosyć rzadko mówi się o tym jaki jest sens przyjętych oznaczeń liczbowych. Co oznaczają kolejne liczby, które opisują schemat próbkowania? W odpowiedzi na to pytanie pomocny okaże się rysunek 1-18.



Rysunek 1-18 Schematy próbkowania chrominancji – interpretacja sensu stosowanych oznaczeń liczbowych.

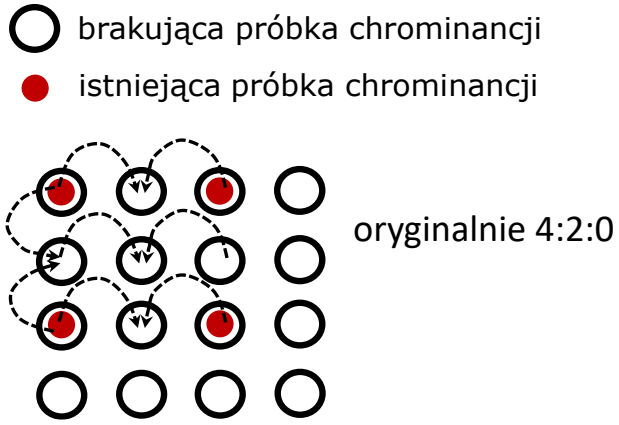
Dla danego schematu próbkowania informacji o kolorze należy dokładnie określić, jaka liczba próbek składowych barwnych C_B oraz C_R przypada na fragment obrazu o zdefiniowanym wcześniej rozmiarze. Najczęściej tym fragmentem jest **region próbek** o szerokości **4 próbki**, który składa się z dwóch wierszy próbek, tak jak zostało to przedstawione na powyższym rysunku. Właśnie stąd bierze się pierwsza cyfra oznaczenia schematu próbkowania (**4**:2:0 w rozważanym przykładzie). Kolejna, druga cyfra określa liczbę próbek każdej składowej koloru (czyli niezależnie C_B i C_R) w pierwszym wierszu rozważanego regionu próbek (dwie próbki każdej składowej w rozważanym przykładzie, czyli 4:**2**:0), a trzecia cyfra liczbę próbek składowych koloru w drugim wierszu regionu (zero próbek składowej C_B i C_R w analizowanym przykładzie, czyli 4:2:**0**). Przedstawiona tutaj konwencja opisu sposobu próbkowania koloru nie mówi, jakie jest rozmieszczenie tych próbek w wierszu pierwszym i drugim regionu, jednak zakłada się tutaj rozłożenie równomierne próbek.

1.4.9. Powrót z reprezentacji YC_BC_R do przestrzeni RGB

Jak już powiedziano wcześniej, reprezentacja YC_BC_R stosowana jest na potrzeby właściwej kompresji obrazu. Samo wyświetlanie obrazu, oryginalnego bądź zdekodowanego, jest realizowane w ramach przestrzeni RGB. Dlatego, po kompresji i dekompresji obrazu jego projekcja musi być poprzedzona powrotnym przejściem z przestrzeni współrzędnych transmisyjnych (YC_BC_R) do układu składowych R, G i B. Składowe R, G, B wylicza się ze składowych Y, C_B, C_R stosując proste kroki obliczeniowe:

$$\begin{gathered} R = Y + 1{,}402 \cdot (C_R - 128) \\ G = Y - 0{,}344136 \cdot (C_B - 128) - 0{,}714136 \cdot (C_R - 128) \\ B = Y + 1{,}772 \cdot (C_B - 128) \end{gathered} \tag{1.3}$$

Zastosowanie powyższych równań wymaga jednak tej samej przestrzennej rozdzielczości składowych koloru C_B, C_R oraz składowej Y. Dlatego przed użyciem równań 1.3 należy przeprowadzić **nadpróbkowanie** (czyli zwiększenie częstotliwości próbkowania sygnału) składowych koloru, jeśli częstotliwość próbkowania C_B i C_R jest mniejsza niż w przypadku składowej Y. Jednym z najprostszych sposobów wyznaczenia wartości brakujących próbek dla składowych jest policzenie średniej z kilku (np. dwóch) próbek, które sąsiadują z próbkami brakującymi i przyjęcie wyników obliczeń jako wartości tych brakujących próbek (patrz rysunek 1-19). Czyli brakujące próbki koloru są wyznaczane w drodze **interpolacji** składowych C_B i C_R. W praktyce próbki brakujące wyznacza się w oparciu o wartości próbek istniejących, bądź próbek wyliczonych już wcześniej.



Rysunek 1-19 Ilustracja przykładowego sposobu wyznaczenia brakujących próbek chrominancji na podstawie próbek istniejących (lub wyliczonych wcześniej). W przedstawionym przykładzie brakujące próbki koloru są wyliczane poprzez uśrednienie dwóch sąsiednich próbek (istniejących lub wcześniej wyznaczonych).

1.5. Obraz ruchomy, czyli cyfrowa sekwencja wizyjna

Za pomocą pojedynczego obrazu można przedstawić jedynie statyczną scenę, czyli taką, której treść nie zmienia się w czasie. Odwzorowanie sceny zmiennej w czasie wymaga użycia sekwencji (ciągu) obrazów, w której każdy z obrazów przedstawia położenie obiektów sceny w ściśle określonej chwili czasowej. Taka sekwencja obrazów cyfrowych jest nazywana **obrazem ruchomym** lub po prostu **cyfrową sekwencją wizyjną**.

Rysunek 1-20 Obraz ruchomy nazywany również cyfrową sekwencją wizyjną. Kolejne obrazy sekwencji przedstawiają treść sceny, jaka została zarejestrowana kamerą w danej chwili czasowej.

Z uwagi na ten przyjęty w technice sposób reprezentacji ruchomego obrazu (właśnie jako ciąg obrazów nieruchomych) wszystkie przedstawione wcześniej metody reprezentacji cyfrowego obrazu nieruchomego mają również zastosowanie w przypadku sekwencji wizyjnej. Na przykład, każdy z obrazów sekwencji przedstawiany jest przy pomocy składowych R, G, B lub składowych Y, C_B, C_R.

Z punktu widzenia widza, który ogląda ruchomy obraz, istotnymi parametrami, które mają wpływ na postrzeganą jakość tego obrazu są: **przestrzenna rozdzielczość** poszczególnych obrazów sekwencji, **liczba bitów** przy pomocy której reprezentujemy każdą **próbkę** kolorowego obrazu, ale również **rozdzielczość czasowa** sekwencji. Ostatni parametr określa liczbę obrazów, które występują w czasie jednej sekundy zarejestrowanej sekwencji. Znaczenie dwóch pierwszych parametrów (rozdzielczość przestrzenna obrazu oraz liczba bitów na próbkę obrazu) dla jakości wyświetlanego obrazu jest takie, jak w przypadku obrazu nieruchomego – stosowne wytłumaczenie zostało już przedstawione w punktach dotyczących próbkowania oraz kwantyzacji obrazu. Parametr trzeci, czyli rozdzielczość czasowa sekwencji wpływa na „płynność” wyświetlania sekwencji wizyjnej, co bezpośrednio przekłada się na wrażenie „płynności” ruchu poszczególnych obiektów zarejestrowanej sceny. Jeśli ta rozdzielczość jest zbyt niska (poniżej 15-20 obrazów na sekundę), to widz ma (lub może mieć) wrażenie, że obiekty sceny poruszają się w sposób skokowy. Żeby tak nie było, konieczne jest rejestrowanie więcej niż 20 obrazów na sekundę. I tak dla przykładu, w stosowanych w Europie i USA systemach telewizji standardowej rozdzielczości przyjęto wartości odpowiednio 25 i 30[9] obrazów na sekundę. Jednak w przypadku najnowszych systemów telewizji oferujących wyższą niż telewizja standardowa, jakość ruchomego obrazu te wartości są znacznie większe i wynoszą: do 60[10] obrazów na sekundę w przypadku telewizji HDTV (ang. High-Definition Television) oraz do 120 obrazów na sekundę, jeśli mowa jest o telewizji UHDTV (ang. Ultra-High-Definition Television).

[9] W przypadku amerykańskiego systemu NTSC jest to dokładnie 29,97 obrazów na sekundę.
[10] Rzeczywista wartość wynosi 59,94.

1.6. Podsumowanie

Ewolucja mikroprocesorów i układów scalonych, połączona z postępem, jaki dokonał się w zakresie metod przetwarzania sygnałów spowodowały zastąpienie obrazu analogowego obrazem cyfrowym. Istotnymi parametrami cyfrowego obrazu są: liczba próbek, z jakiej składa się pojedynczy obraz, liczba bitów, którą przeznaczamy na reprezentację każdej próbki obrazu, oraz liczba obrazów, która reprezentuje jedną sekundę sekwencji. Wymienione parametry bezpośrednio wpływają na wrażenie jakości oglądanego przez widza obrazu, obrazu ruchomego i obrazu nieruchomego, o czym można się było przekonać studiując zamieszczony w tym rozdziale opis.

Punktem startowym dla właściwej kompresji obrazu jest jego reprezentacja przestrzenna. W tym rozdziale skupiono się tylko na takich sposobach reprezentacji przestrzennej obrazu, które mają szczególne znaczenie właśnie z perspektywy kompresji obrazu i jego wyświetlania. Od samego początku ery kolorowego obrazu w telewizji, takimi przestrzeniami są reprezentacje **RGB** oraz **YC_BC_R**, i to one były tematem przeprowadzonych w tym rozdziale rozważań. Z przedstawionego powyżej powodu autor zrezygnował z omawiania innych, dobrze w literaturze znanych sposobów reprezentacji obrazów cyfrowych, które nie mają jednak bezpośredniego zastosowania w kompresji danych obrazowych, jak np. mająca zastosowanie w grafice przestrzeń **HSV** – Hue, Saturation, Value, czy stosowana w praktyce drukarskiej reprezentacja **CMYK** – Cyan, Magenta, Yellow, Key Colour. Jednak o tych przestrzeniach barw, oraz innych tutaj niewymienionych znaleźć można informacje w innych opracowaniach, np. w książce [Doma10].

Właściwa idea opisująca reprezentacje RGB i YC_BC_R nie zmieniła się na przestrzeni lat. Należy jednak zauważyć zmiany wartości określonych parametrów tych przestrzeni. Dokonujący się postęp technologiczny w zakresie kamer oraz wyświetlaczy obrazu stworzył możliwości praktycznego zastosowania odmian przestrzeni RGB o szerszym niż było to pierwotnie, zakresie kolorów. Dobrym przykładem jest stworzona na potrzeby telewizji wysokiej jakości norma BT.2020. Możliwość odwzorowania szerszego spektrum barw w obrazie, w połączeniu ze zwiększonym zakresem wartości, jakie może przyjąć każda próbka składowych obrazu (czyli w efekcie większa rozpiętość pomiędzy poziomem jasności próbki jasnej i ciemnej w obrazie danej składowej plus większa liczba poziomów jasności w ogóle[11]), dają w rezultacie obraz, który w dużo wierniejszy sposób oddaje rzeczywistą treść sceny. Obok wymienionych wcześniej zmian parametrów, faktem jest również rejestracja i kompresja obrazów o coraz wyższej rozdzielczości przestrzennej oraz czasowej (dotyczy przestrzeni RGB i YC_BC_R). Efektem przytoczonych zmian jest nowa, niespotykana wcześniej jakość cyfrowego obrazu, który w dokładniejszy sposób przybliża oglądany świat rzeczywisty. Ceną, którą płacimy za ten postęp jest rosnąca objętość oryginalnych danych, które te obrazy reprezentują. Dlatego, wraz z tym postępem, wzrasta również znaczenie technik kompresji danych obrazowych, które otwierają drogę dla rynkowego wdrożenia nowych technologii.

[11] Mowa jest tutaj o technologii HDR – High Dynamic Range. Technologia HDR pozwala na odwzorowanie dużo większego zakresu jasności punktów obrazu, dając możliwość ukazania szczegółów treści obrazu zarówno w jego fragmentach ciemnych, jak i jasnych.

Rozdział 2

Częstotliwościowa reprezentacja obrazu

2.1. Wprowadzenie

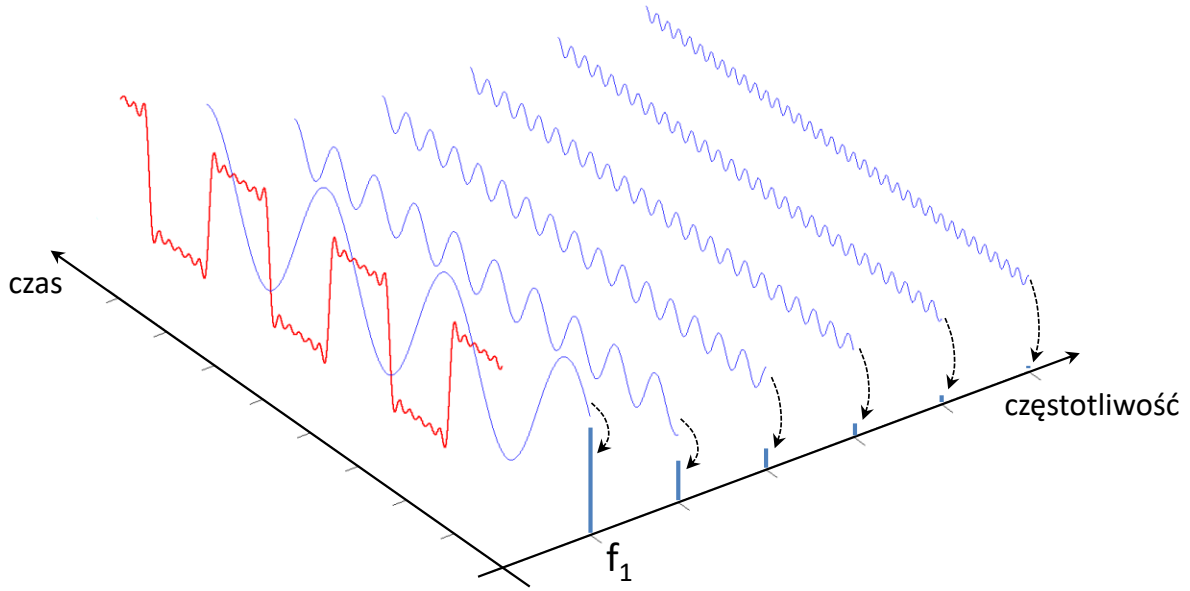
Bezpośrednie kodowanie wartości próbek składowych Y, C_B, C_R obrazu jest mało wydajne, i w niewielkim tylko stopniu redukuje rozmiar oryginalnego strumienia danych, który ten obraz opisuje. Tę niewielką wydajność kompresji można dość istotnie zwiększyć stosując inne metody kompresji, np. takie, które uwzględniają fakt silnego podobieństwa wartości sąsiednich próbek w obrazie[12]. W przytoczonym przykładzie, zamiast niezależnego kodowania kolejnych próbek obrazu można skutecznie (czyli z dużą dokładnością) przewidywać ich wartości na podstawie próbek sąsiednich i kodować już sam błąd przewidywania próbek, czyli sygnał błędu predykcji, który zwykle przyjmuje bardzo małe wartości. Jednak nawet **technika predykcyjnego kodowania obrazu**, bo o niej tutaj mowa, nie sprawdzi się dobrze we wszystkich zastosowaniach, np. nie umożliwi wydajnej, stratnej kompresji obrazu, która byłaby realizowana z uwzględnieniem faktycznych ograniczeń percepcji obrazu przez człowieka. W związku z tym potrzebne są jeszcze inne sposoby kodowania obrazu, dla których technika kodowania predykcyjnego będzie swego rodzaju uzupełnieniem.

Jednym z takich sposobów, którego znaczenie w kompresji jest wręcz ogromne jest kodowanie danych obrazowych w tzw. **dziedzinie częstotliwości obrazu**. U podstaw tej metody leżą naukowe odkrycia Eulera i Fouriera[13], zgodnie z którymi każdą funkcję (sygnał) można przedstawić jako sumę funkcji typu sinusoidalnego (czyli funkcje sinus, czy kosinus)[14], nazywanych w literaturze **składowymi harmonicznymi**. Żeby dany sygnał móc w ten sposób reprezentować, należy do wspomnianej sumy wziąć ściśle określony zbiór składowych sinusoidalnych lub kosinusoidalnych, czyli przebiegi o ściśle określonej **amplitudzie**, **częstotliwości** oraz przesunięciu na osi argumentów, za które odpowiada **faza początkowa** (można też mówić o **przesunięciu fazowym**). W takiej reprezentacji sygnału składowe harmoniczne są niczym „klocki” przy pomocy których budowany jest sygnał. Tym sygnałem może być również obraz.

Przedstawioną ideę **częstotliwościowej reprezentacji sygnału** dobrze ilustruje zamieszczony na rysunku 2-1 przykład, w którym okresowy przebieg, który przypomina falę prostokątną, jest opisywany poprzez sumę składowych harmonicznych, z których każda ma konkretną wartość **amplitudy**, **częstotliwości** oraz **fazy początkowej**. W przypadku sygnału okresowego (czyli takiego, którego wartości cyklicznie się powtarzają), częstotliwości

[12] Z całą pewnością tak jest w przypadku obrazów, które przedstawiają sceny naturalne.
[13] Wspomniane prace zostały zapoczątkowane w wieku XVIII i na początku wieku XIX.
[14] Nie ma znaczenia, czy użyjemy funkcji sinus, czy też kosinus. Funkcje te różnią się tylko fazą początkową.

poszczególnych składowych harmonicznych są zawsze całkowitymi wielokrotnościami częstotliwości f_1 tzw. podstawowej (czyli pierwszej) składowej harmonicznej, co ma bardzo duże znaczenie w praktyce kompresji sygnałów. W opisywany tutaj sposób daje się przedstawić dowolny inny sygnał, należy tylko użyć właściwego zbioru składowych harmonicznych. Tak więc, dzięki reprezentacji częstotliwościowej, zamiast bezpośredniego kodowania informacji o tym, jakie są wartości sygnału dla kolejnych wartości argumentu (w przypadku sygnału z rysunku 2-1 argument definiuje kolejne chwile czasowe) prześlemy do dekodera jedynie parametry składowych harmonicznych (amplituda, częstotliwość, faza początkowa), z których składa się nasz sygnał. Ponieważ ogólna postać składowych harmonicznych jest nam dobrze znana, zakodowanie samych parametrów tych składowych wymagać będzie w ogólności dużo mniejszej liczby bitów, niż ma to miejsce w przypadku bezpośredniego kodowania kolejnych wartości sygnału. Dodatkowo, reprezentując nasz sygnał w dziedzinie częstotliwości mamy bezpośrednią możliwość uwzględnienia w trakcie kodowania znanych nam ograniczeń aparatu widzenia człowieka i kodować z różną (a nie taką samą) dokładnością składowe harmoniczne nisko- i wysokoczęstotliwościowe. Stąd olbrzymie wręcz znaczenie wyników prac Eulera i Fouriera w teorii oraz praktyce przetwarzania i kompresji sygnałów, w tym sygnałów, które reprezentują obrazy.



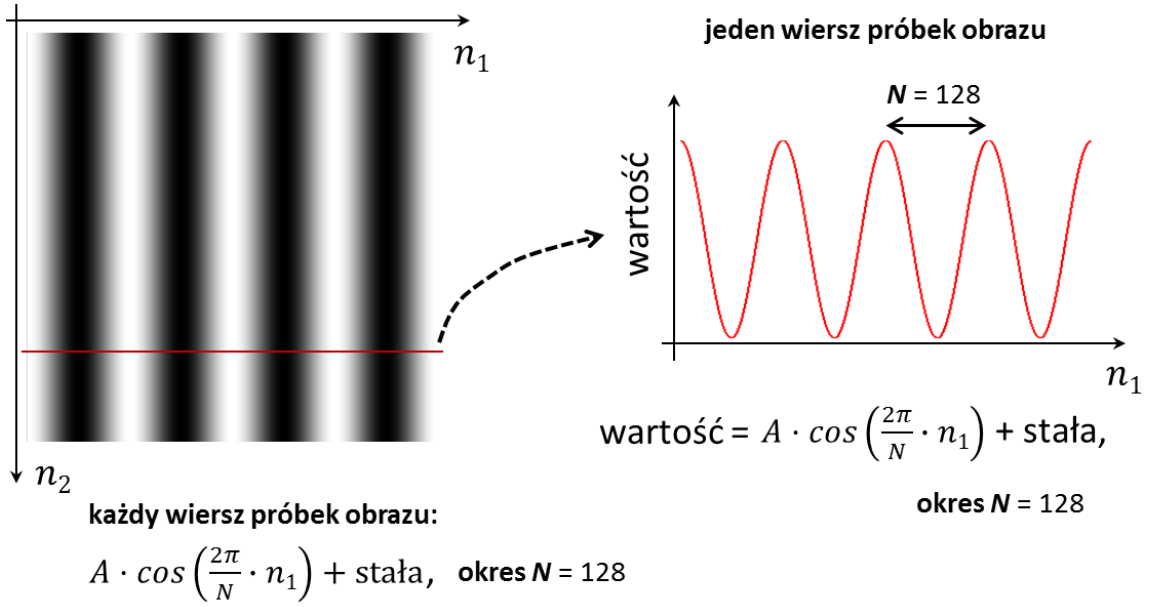
Rysunek 2-1 Ilustracja idei **częstotliwościowej reprezentacji sygnału**. Okresowy przebieg przypominający prostokątny (oznaczony czerwoną linią) jest wynikiem sumy przebiegów sinusoidalnych (składowych harmonicznych) o określonej amplitudzie, częstotliwości i fazie początkowej. Idea wykonania rysunku została zaczerpnięta z materiałów zamieszczonych na stronie internetowej http://www.stevejtrettel.com/fourier-series--transform.html.

2.2. Obraz i jego reprezentacja częstotliwościowa – podstawy

W poprzednim punkcie, idea reprezentacji częstotliwościowej sygnału została zilustrowana z perspektywy sygnału jednowymiarowego, czyli sygnału będącego funkcją tylko jednej zmiennej niezależnej. Jednak przy pomocy składowych harmonicznych możemy przedstawiać także bardziej skomplikowane sygnały, o większej liczbie wymiarów, czyli również obrazy. W przypadku obrazów składowe harmoniczne pozwalają na odwzorowanie zmian wartości próbek (próbek, które opisują treść obrazu) o okresowym charakterze. Żeby lepiej zrozumieć tę ideę, przeanalizujmy zamieszczone w kolejnych punktach przykłady.

2.2.1. Zmiany treści w kierunku poziomym, czyli „częstotliwość przestrzenna pozioma”

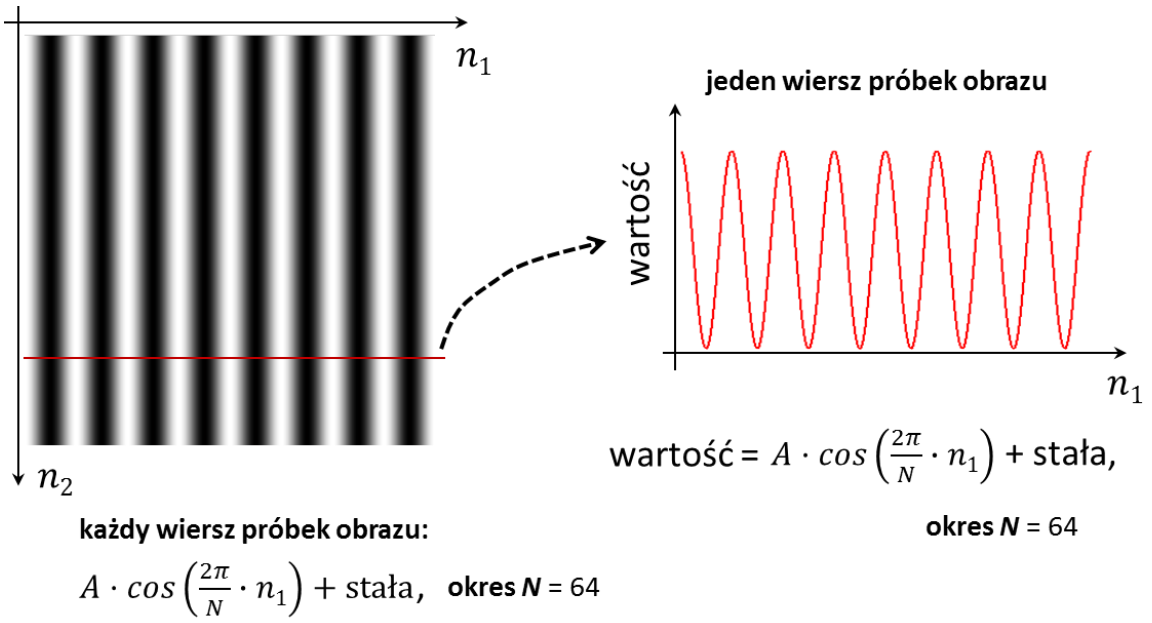
Na początku wyobraźmy sobie obraz, którego postać jest taka jak na rysunku 2-2. W tym szczególnym obrazie **wartości próbek w każdym wierszu próbek okresowo się zmieniają i tworzą funkcję kosinusoidalną o ściśle określonej częstotliwości** $f = \frac{1}{N}$ (częstotliwość ta przekłada się na okres *N* zmian wartości próbek obrazu). Każdy wiersz próbek tego obrazu można więc przedstawić przebiegiem kosinusoidalnym o danej wartości częstotliwości (oraz danej wartości minimalnej i maksymalnej funkcji kosinus). Taki sposób reprezentacji obrazu daje wymierne korzyści, ponieważ w omawianym przypadku, zamiast kodować wartość każdej próbki obrazu niezależnie (co oczywiście wiązałoby się z niemałym kosztem bitowym) możemy wysłać do dekodera „krótką” wiadomość, która mówi, że kolejne próbki w każdym wierszu próbek obrazu mają wartości takie jak wartości funkcji kosinusoidalnej o zadanych parametrach (określona wartość minimalna, pewna wartość maksymalna funkcji oraz częstotliwość zmian wartości). Do dekodera wysłalibyśmy zatem tylko parametry tej kosinusoidy, dzięki której obliczylibyśmy wartości kolejnych próbek w każdej linii próbek obrazu.



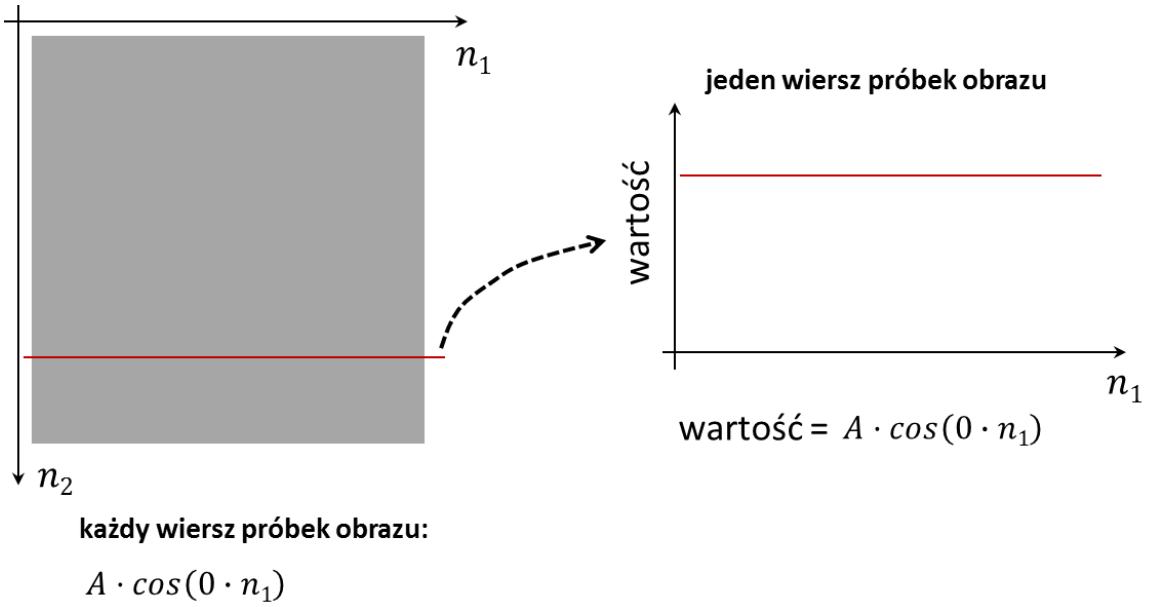
Rysunek 2-2 Przykładowy obraz (po lewej), w którym wartości próbek w każdym wierszu próbek zmieniają się zgodnie z funkcją kosinusoidalną. Każdy wiersz próbek obrazu można więc przedstawić funkcją kosinusoidalną o ściśle określonej częstotliwości $f = \frac{1}{N}$ (o ściśle określonym okresie *N*, który jest wyrażany w próbkach obrazu).

Gdybyśmy w stosunku do obrazu z rysunku 2-2 zwiększyli dwukrotnie częstotliwość zmian wartości próbek w każdym wierszu, to otrzymalibyśmy obraz taki jak na rysunku 2-3. Zgodnie z poczynioną tutaj zmianą, wartości kolejnych próbek w linii również się okresowo zmieniają, z tą różnicą, że zmiana ta następuje dwa razy szybciej niż w poprzednio analizowanym przykładzie. Wartości próbek w każdej linii próbek „układają” się zatem w trochę inną funkcję kosinusoidalną, której częstotliwość jest dwa razy większa, w porównaniu do poprzedniej funkcji. Właściwe „oddanie” tej szybszej zmiany wartości próbek wymaga użycia składowej harmonicznej sygnału o odpowiednio wysokiej wartości częstotliwości. **Tym samym dochodzimy w tym miejscu do**

następującego ważnego wniosku: częste (szybkie) zmiany wartości próbek uda się odwzorować tylko funkcjami harmonicznymi o wysokiej częstotliwości, podczas gdy do odwzorowania próbek obrazu, których wartości zmieniają się „powoli” potrzebne będą składowe harmoniczne wolnozmienne, czyli o małej wartości częstotliwości. W skrajnym przypadku obrazu, w którym częstotliwość zmian wartości próbek jest zerowa (czyli wszystkie próbki mają taką samą wartość – patrz rysunek 2-4) do reprezentacji każdego wiersza próbek użyjemy składowej o zerowej wartości częstotliwości, czyli wartości stałej (nazywanej po prostu składową stałą).



Rysunek 2-3 Przykład innego obrazu (po lewej), w którym wartości próbek w każdym wierszu próbek zmieniają się również zgodnie z funkcją kosinusoidalną. W porównaniu z zamieszczonym na rysunku 2-2 przykładzie częstotliwość zmian wartości próbek w każdym wierszu jest tutaj dwa razy większa (czyli okres *N* tych zmian jest dwukrotnie krótszy).

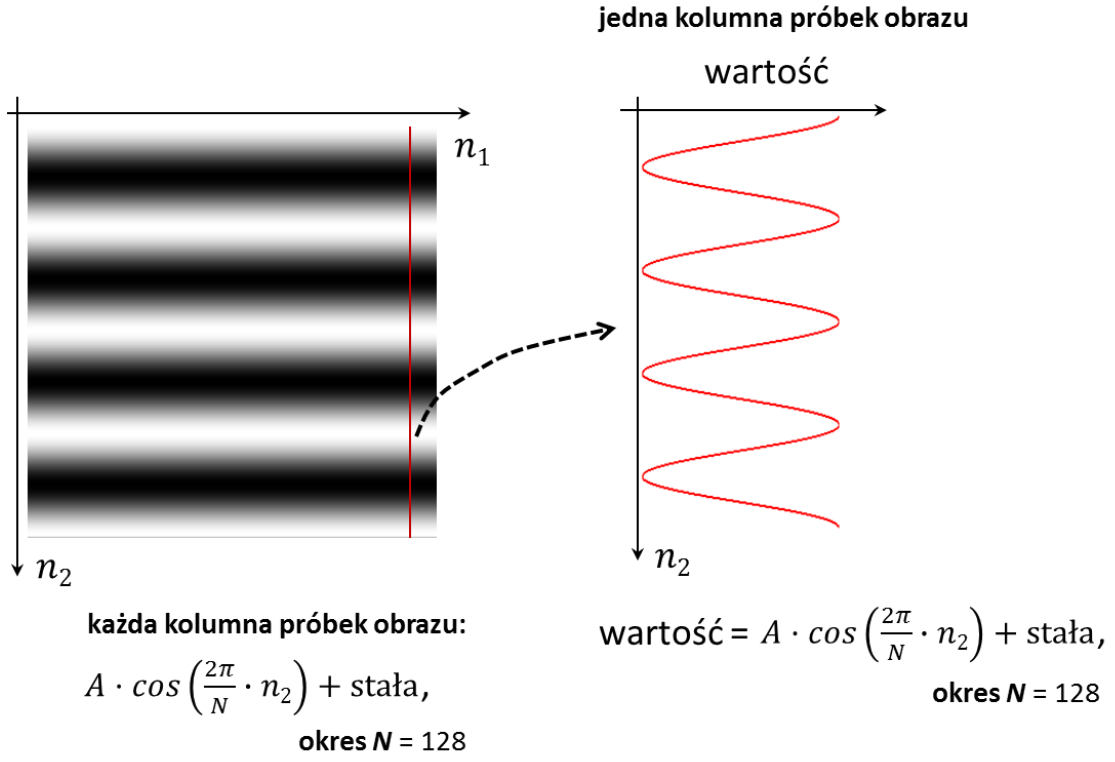


Rysunek 2-4 Obraz (po lewej), w którym wszystkie próbki w każdym wierszu mają taką samą wartość. Częstotliwość zmian wartości próbek jest zatem zerowa (w każdym wierszu).

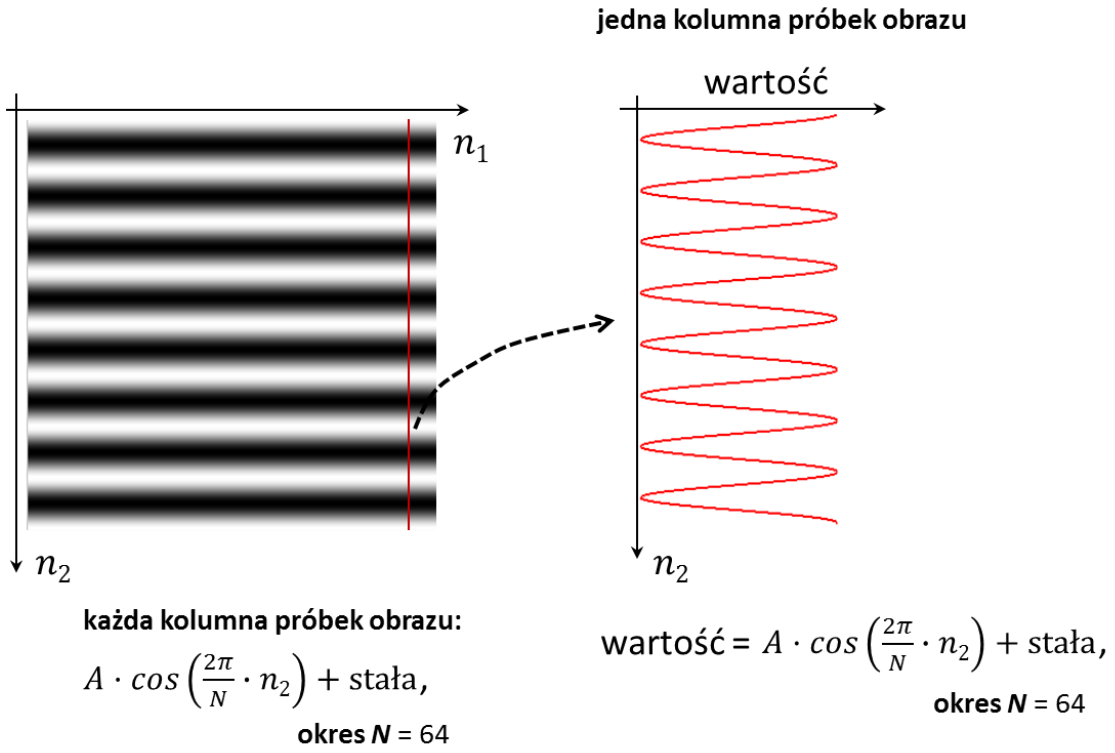
W przedstawionych na rysunkach 2-2 i 2-3 przykładach przebiegi kosinusoidalne odwzorowują charakter zmian wartości próbek w każdej linii próbek, w tym również **częstotliwość** f tych **zmian**. Ponieważ w przedstawionych wcześniej przykładach zmiany wartości próbek obrazu dokonywały się w kierunku poziomym w obrazie, to częstotliwość f tych zmian jest nazywana **częstotliwością przestrzenną poziomą**.

2.2.2. Zmiany treści w kierunku pionowym, czyli „częstotliwość przestrzenna pionowa”

Jednak tradycyjny obraz ma dwa wymiary, w związku z czym zmiany wartości próbek mogą się również dokonywać w kierunku pionowym w obrazie. Na dwóch kolejnych rysunkach przedstawiono przykłady takich obrazów. Podobnie jak miało to miejsce wcześniej, przedstawione na rysunkach 2-5 i 2-6 obrazy różnią się częstotliwością zmian wartości próbek, gdyby analizować te wartości w każdej kolumnie próbek (czyli właśnie w kierunku pionowym). Tak samo jak w przykładach z rysunków 2-2 i 2-3 „prędkość” zmian wartości próbek determinuje częstotliwość składowej harmonicznej, z użyciem której możemy te zmiany reprezentować.



Rysunek 2-5 Przykładowy obraz (po lewej), w którym wartości próbek w każdej kolumnie próbek zmieniają się zgodnie z funkcją kosinusoidalną. Każdą kolumnę próbek obrazu można więc przedstawić funkcją kosinusoidalną, o ściśle określonej częstotliwości $f = \frac{1}{N}$ (o ściśle określonym okresie *N*, który jest wyrażany w próbkach obrazu).

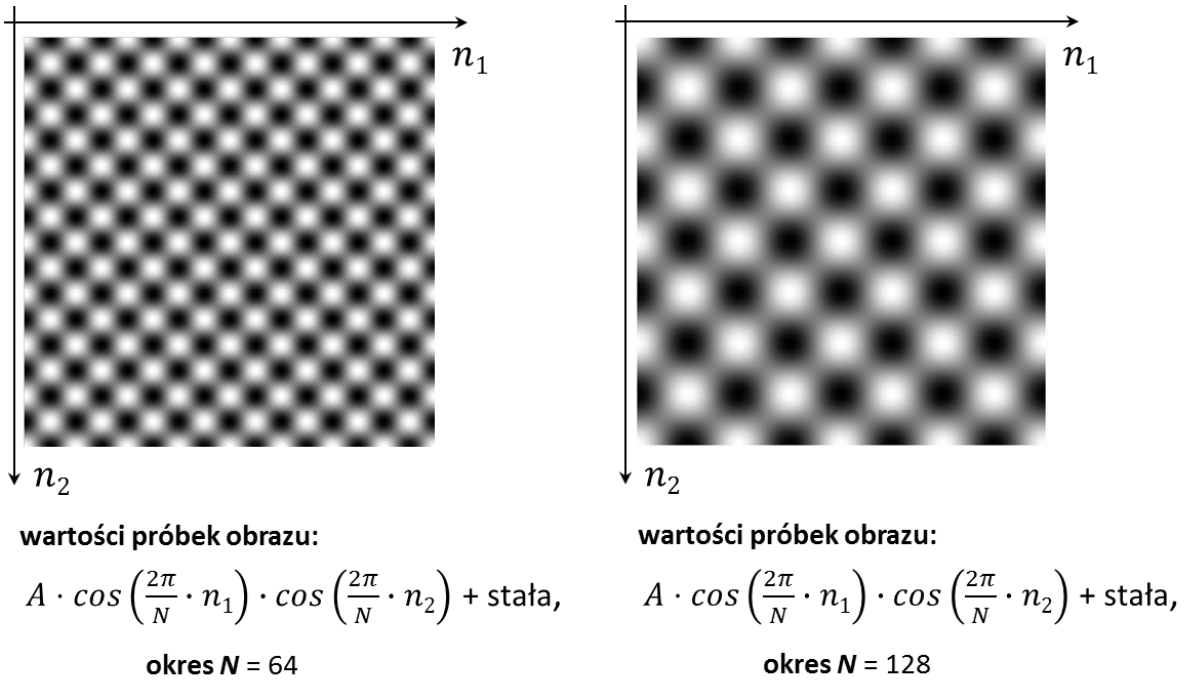


Rysunek 2-6 Przykład innego obrazu (po lewej), w którym wartości próbek w każdej kolumnie próbek zmieniają się również zgodnie z funkcją kosinusoidalną. W porównaniu z zamieszczonym na rysunku 2-5 przykładzie częstotliwość zmian wartości próbek w każdej kolumnie jest tutaj dwa razy większa (czyli okres tych zmian jest dwukrotnie krótszy).

2.2.3. Bardziej rzeczywisty przypadek, czyli zmiany w obu kierunkach naraz

W przedstawionych do tej pory przykładach obserwowaliśmy okresowe zmiany treści obrazu, które dokonywały się tylko w jednym z dwóch wskazanych kierunków w obrazie: poziomym lub pionowym. Zmiany te opisywaliśmy pewną funkcją harmoniczną (czyli typu sinusoidalnego), która „biegła” w jednym kierunku (poziomym lub pionowym). Bardzo często spotkamy się jednak z sytuacją, w której zmiany treści będą występować w obu kierunkach naraz. Prostym przykładem takiej sytuacji są dwa obrazy, które zostały przedstawione na rysunku 2-7.

Żeby pokazane zmiany treści obrazów móc zaprezentować potrzebne są jednocześnie dwie funkcje harmoniczne, z których jedna opisze zmiany wartości próbek zachodzące w kierunku poziomym (czyli wzdłuż zmiennej n_1), a druga w kierunku pionowym (wzdłuż zmiennej n_2). Właściwa „kombinacja” tych funkcji (z matematycznego punktu widzenia jest to operacja iloczynu dwóch funkcji) pozwoli nam opisać zmiany wartości próbek w całym dwuwymiarowym obrazie. Można powiedzieć, że iloczyn dwóch funkcji harmonicznych, z których jedna jest funkcją zmiennej n_1 a druga funkcją zmiennej n_2 tworzy **dwuwymiarową funkcję (składową) harmoniczną**, która pozwala na prezentację treści dwuwymiarowego obrazu.



Rysunek 2-7 Przykłady dwóch obrazów, w których wartości próbek w każdej kolumnie próbek oraz w każdym wierszu próbek zmieniają się zgodnie z funkcją typu sinusoidalnego. Obrazy lewy i prawy różnią się częstotliwością zmian wartości próbek. Każdy z przedstawionych obrazów można matematycznie opisać iloczynem dwóch funkcji kosinusoidalnych, z których jedna jest funkcją zmiennej n_1, a druga funkcją zmiennej n_2. Otrzymany wynik należy jeszcze skorygować dodając do niego pewną wartość stałą (składową stałą).

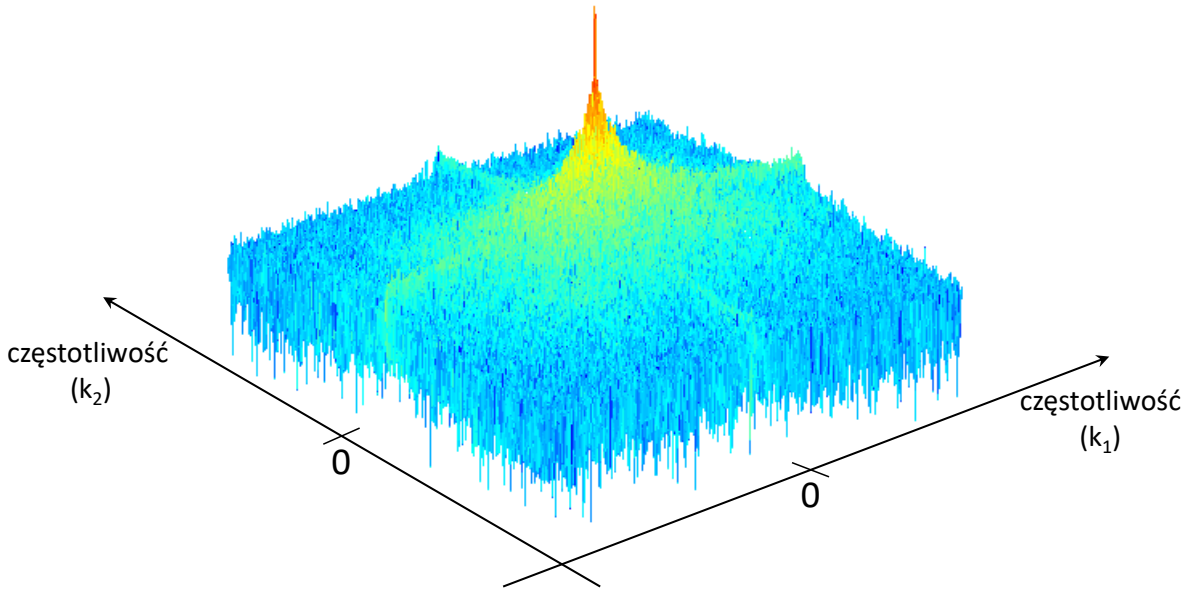
2.3. Częstotliwościowa „zawartość” (reprezentacja) obrazu naturalnego

2.3.1. Obraz i jego widmo amplitudowe

W poprzednich punktach przeprowadzono analizę zawartości częstotliwościowej bardzo prostych obrazów. W tym punkcie zobaczymy, które z częstotliwości przestrzennych poziomych oraz częstotliwości przestrzennych pionowych wchodzą w skład dużo bardziej złożonych obrazów, które reprezentują sceny naturalne (czyli sceny zarejestrowane aparatem lub kamerą). Przedmiotem analizy będzie zamieszczony na rysunku 2-8 obraz „Lena”.

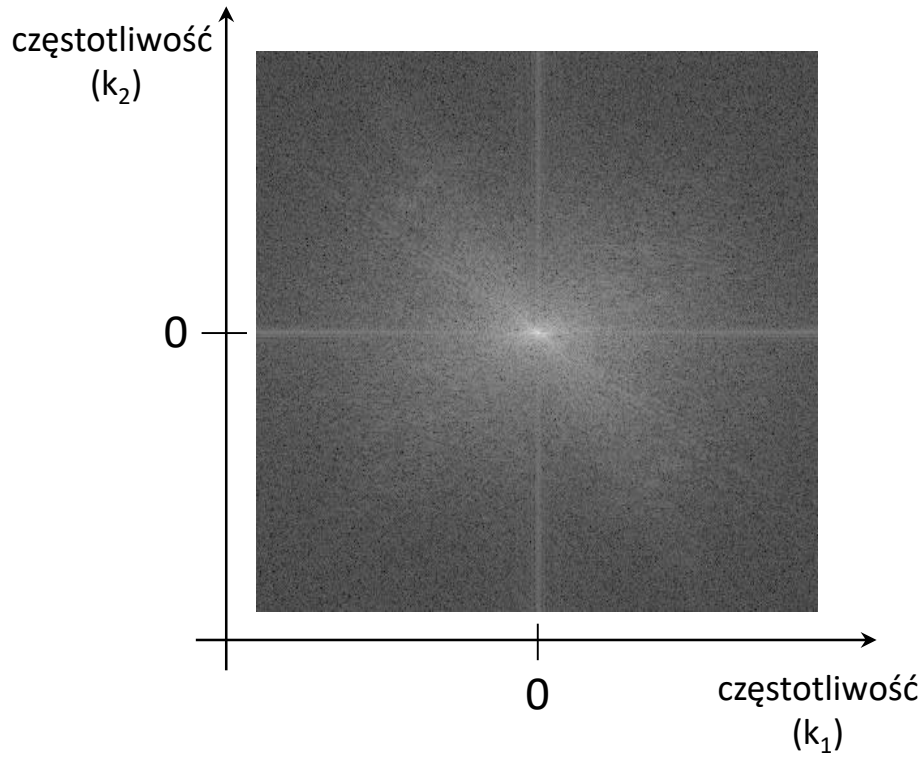
Rysunek 2-8 Wersja monochromatyczna obrazu „Lena”.

Analiza zawartości częstotliwościowej tego obrazu prowadzi do częściowego wyniku, który został przedstawiony na rysunku 2-9. Dla wygody czytelnika ten sam wynik został dodatkowo przedstawiony w nieco innej konwencji – zobacz rysunek 2-10.



Rysunek 2-9 „Zawartość” częstotliwościowa obrazu „Lena”. Wykres przedstawia **widmo amplitudowe** tego obrazu (ale wyrażone w skali logarytmicznej). Dla większej czytelności wykresu dane zostały przedstawione w skali logarytmicznej, obliczając wartości jako $100 \cdot log_{10}(1 + |F|)$, gdzie F jest **widmem** obrazu „Lena” obliczonym za pomocą dwuwymiarowego dyskretnego przekształcenia Fouriera, a $|F|$ to **widmo amplitudowe** obrazu[15].

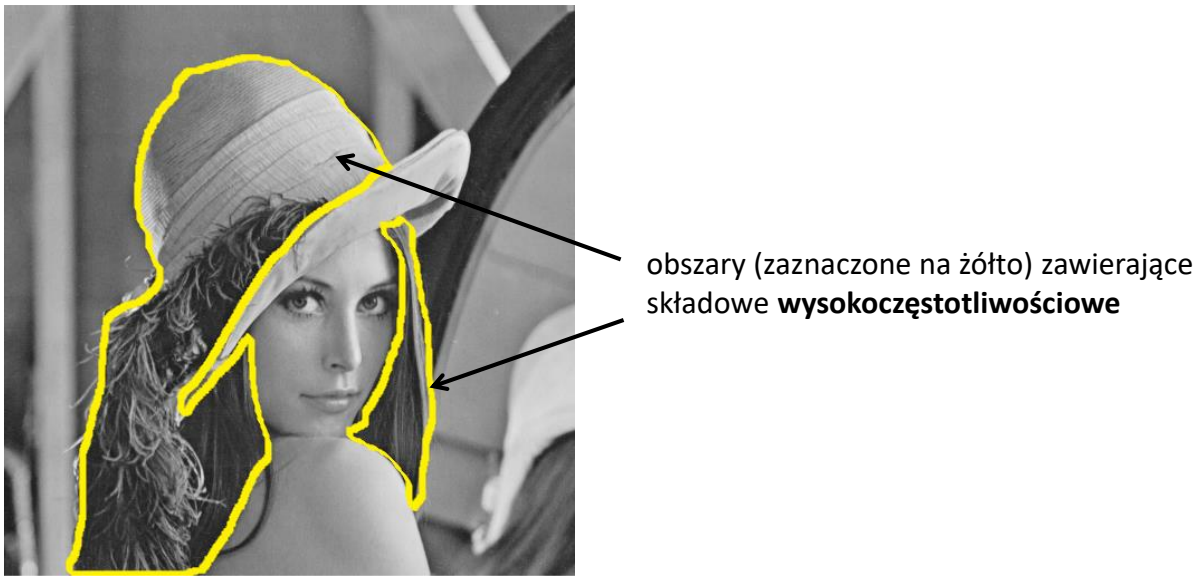
[15] Widmo sygnału dyskretnego jest okresowe wzdłuż obu osi częstotliwości. Rysunek przedstawia zatem tylko jeden cykl okresowego widma obrazu.



Rysunek 2-10 „Zawartość” częstotliwościowa obrazu „Lena”. Wykres przedstawia „widok” **widma amplitudowego** widziany z góry. Poziomem jasności wyrażona została wartość amplitudy składowej harmonicznej o danej częstotliwości, zgodnie z regułą: im jaśniejszy punkt, tym większa wartość amplitudy składowej. Widmo amplitudowe wyznaczono w sposób przedstawiony w opisie rysunku 2-9.

Powyższe wykresy przedstawiają **widmo amplitudowe** obrazu „Lena”, czyli dają odpowiedź na następujące pytanie: składowe harmoniczne o jakiej wartości częstotliwości (częstotliwości przestrzennej poziomej k_1 oraz częstotliwości przestrzennej pionowej k_2) wchodzą w skład analizowanego obrazu? I dodatkowo, jaka jest wartość amplitudy poszczególnych składowych harmonicznych (czyli składowych o konkretnej wartości częstotliwości k_1 i k_2)? Na podstawie otrzymanego wyniku analizy można sformułować ważny wniosek, że największą wartość amplitudy posiadają niskoczęstotliwościowe składowe harmoniczne, czyli składowe, które są skupione wokół częstotliwości o zerowej wartości. Składowe wysokoczęstotliwościowe również występują w obrazie naturalnym (tak jest w ogólności), jednak ich amplitudy są dużo mniejsze w porównaniu z amplitudami składowych harmonicznych o częstotliwościach niskich. Jednak w tym miejscu można zadać sobie pytanie: w jaki sposób otrzymany wynik przekłada się na treść analizowanego obrazu?

Odpowiedź na to pytanie już padła przy okazji dyskusji wyników analizy częstotliwościowej prostych obrazów (patrz punkt 2.2), jednak dla większego porządku zostanie ona w tym miejscu raz jeszcze powtórzona. Jeśli w obrazie mamy fragmenty gładkie, pozbawione szybkich zmian treści to do ich właściwej reprezentacji „wystarczą” same składowe harmoniczne wolnozmienne, czyli składowe o niskiej wartości częstotliwości. W przypadku skomplikowanych obszarów obrazu, w których dochodzi do szybkiej zmiany wartości próbek obrazu (czyli są to obszary z dużą liczbą krawędzi, detali), konieczne jest użycie wysokoczęstotliwościowych składowych harmonicznych, które pozwolą „oddać” te szybkie zmiany treści. Patrząc na treść obrazu „Lena” możemy w nim dosyć łatwo zlokalizować wspomniane dwa typy obszarów: obszary wolno- i szybkozmienne (zobacz poniższy rysunek).



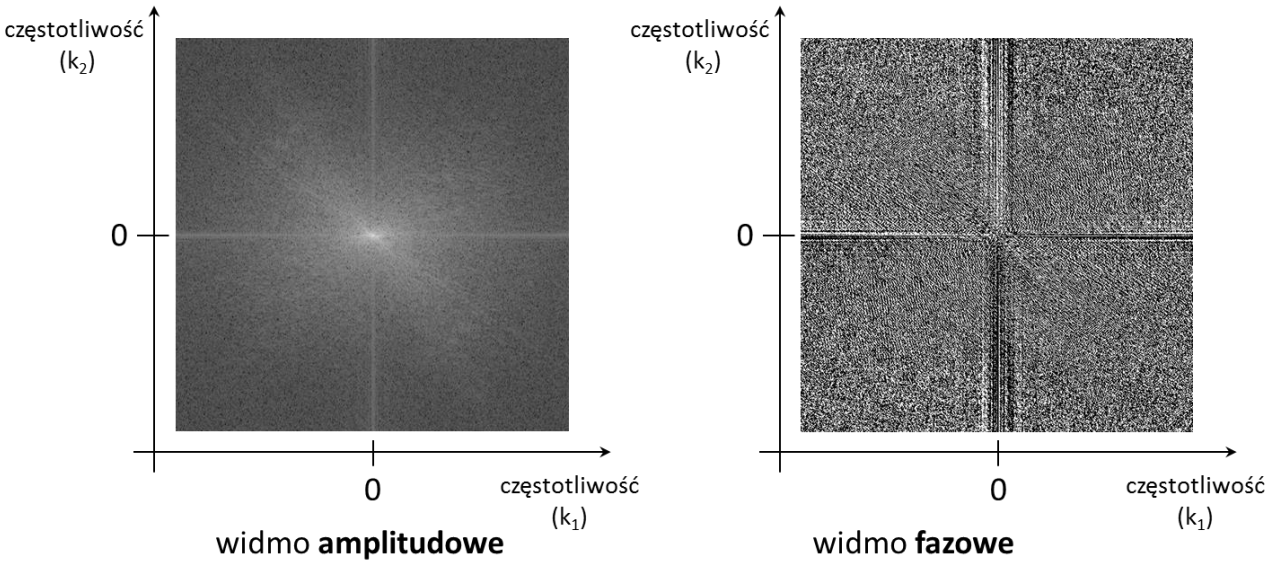
Rysunek 2-11 Linią w kolorze żółtym zaznaczone zostały fragmenty obrazu „Lena”, które zawierają dużą ilość składowych wysokoczęstotliwościowych. Odpowiada to fragmentom o skomplikowanej teksturze, z dużą liczbą krawędzi w tych obszarach. W pozostałych fragmentach obrazu, które są pozbawioną krawędzi gładką teksturą przeważają „częstotliwości niskie”.

Jak dobrze widać, w obrazie „Lena” przeważają obszary relatywnie gładkie, czyli takie, które są pozbawione dużej liczby krawędzi, detali, co jest cechą charakterystyczną obrazów naturalnych. Obszary, które zawierają mocno skomplikowaną treść (zakreślone na powyższym rysunku żółtą linią), czyli z dużą liczbą krawędzi nie pokrywają w związku z powyższym znaczącej części obrazu.

2.3.2. Widmo fazowe obrazu

Przedstawione w poprzednim punkcie **widmo amplitudowe** obrazu stanowi tylko częściowy wynik jego analizy częstotliwościowej. Sama więc wiedza o częstotliwościach składowych harmonicznych wchodzących w skład obrazu, wraz z informacją o amplitudzie poszczególnych składowych, nie umożliwią jeszcze prawidłowego odtworzenia próbek obrazu w dekoderze. I to z bardzo prostego powodu. Ciągle nie będziemy wiedzieć, w których dokładnie częściach obrazu znaleźć się mają obszary o danym charakterze zmian treści (zmiany wolno- bądź szybkozmienne). Dla pozyskania tej wiedzy potrzebujemy zatem czegoś jeszcze…

Tym brakującym elementem jest wiedza o fazie początkowej poszczególnych składowych harmonicznych. Informacja ta jest „przenoszona” przez **widmo fazowe**, którego postać dla testowego obrazu „Lena” została przedstawiona na rysunku 2-12. Dzięki informacji o fazie początkowej składowych, wspomaganej dodatkowo wiedzą o amplitudzie poszczególnych składowych harmonicznych, możliwe się staje pełne odtworzenie treści kolejnych fragmentów obrazu.



Rysunek 2-12 Widmo amplitudowe (po lewej) oraz widmo fazowe (po prawej) obrazu „Lena”. Połączenie obu rodzajów widma daje kompletną wiedzę o częstotliwościowej „zawartości” obrazu. Ta kompletna wiedza umożliwia odtworzenie wartości próbek obrazu.

2.4. Reprezentacja „częstotliwościowa” obrazu w ujęciu matematycznym

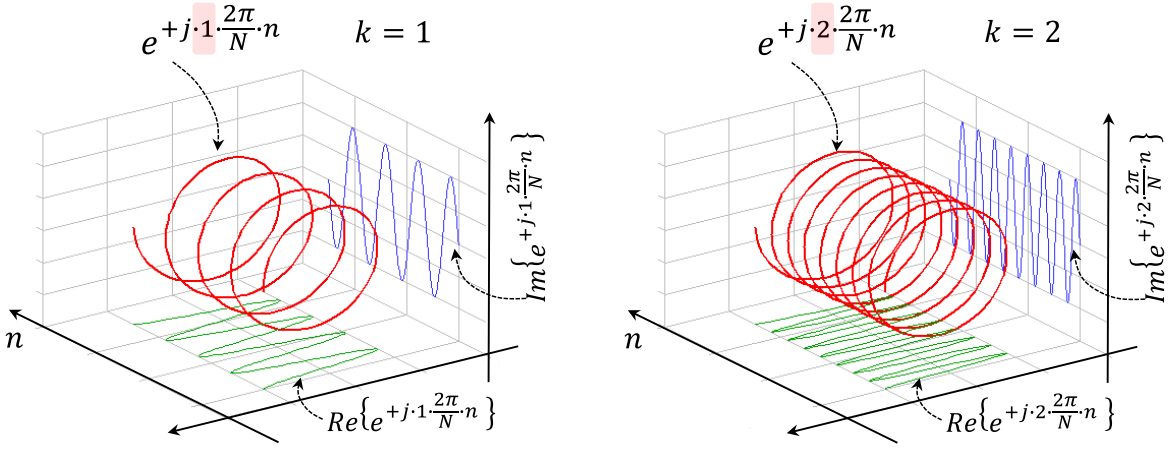
Przedstawioną w poprzednich punktach ideę częstotliwościowej reprezentacji obrazu realizują ściśle określone formuły matematyczne. Ponieważ początkującemu czytelnikowi pełne zrozumienie tych formuł może nastręczać pewne trudności, ich istota zostanie tutaj przedstawiona „krok po kroku”.

Obraz chcemy przedstawić jako **sumę składowych harmonicznych o danej częstotliwości**. Żeby móc w ten sposób reprezentować dowolny sygnał, czyli również taki, który przyjmuje wartości zespolone (czyli wartości zawierające część rzeczywistą oraz część urojoną), musimy użyć **zespolonych składowych harmonicznych**, które są wyrażane poprzez zespoloną funkcję $e^{+jk\cdot\frac{2\pi}{N}\cdot n}$ w której:

- parametr k skaluje częstotliwość podstawowej (czyli pierwszej) składowej harmonicznej (pulsacja[16] tej pierwszej składowej wynosi $\frac{2\pi}{N}$);
- N jest okresem podstawowej, czyli pierwszej składowej harmonicznej. Jej częstotliwość wynosi zatem $\frac{1}{N}$;
- n jest argumentem funkcji składowej harmonicznej, czyli dla danego k zespolona składowa harmoniczna jest okresową funkcją zmiennej n (jej pulsacja wynosi $k \cdot \frac{2\pi}{N}$).

Rysunek 2-13 przedstawia wykres funkcji $e^{+jk\cdot\frac{2\pi}{N}\cdot n}$ dla dwóch różnych wartości parametru k, który współdecyduje o częstotliwości zmian wartości funkcji ($k = 1$ oraz $k = 2$).

[16] Matematyczna zależność pomiędzy pulsacją ω i częstotliwością f składowej jest następująca: $\omega = 2 \cdot \pi \cdot f$.



Rysunek 2-13 Wykres (krzywa czerwona) **zespolonej składowej harmonicznej** $e^{+jk\cdot\frac{2\pi}{N}\cdot n}$ dla dwóch różnych wartości parametru k. Parametr k współdecyduje o częstotliwości zmian wartości składowej harmonicznej.

Analizując postać wykresu zespolonej składowej harmonicznej (czerwona krzywa) możemy sformułować wniosek, iż w omawianym przypadku sygnał (w ogólności o zespolonych wartościach) jest przedstawiany poprzez ważoną sumę „sprężynek” o różnych częstotliwościach. „Sprężynki” te od razu opisują część rzeczywistą jak i część urojoną reprezentowanego sygnału.

Obraz (statyczny) jest jednak sygnałem dwuwymiarowym, dlatego do jego reprezentacji potrzebujemy dwóch „paczek” składowych harmonicznych: składowych, które będą reprezentować zmiany treści obrazu dokonujące się w kierunku poziomym (wzdłuż zmiennej n_1 w obrazie), oraz składowych, które pozwolą na wyrażenie zmian treści zachodzące w kierunku pionowym (wzdłuż zmiennej n_2 w obrazie). Ponieważ częstotliwości zmian treści obrazu mogą być w obu kierunkach – poziomym i pionowym – inne to potrzebujemy dwóch niezależnych indeksów częstotliwości: k_1 i k_2. W ten sposób dochodzimy do konieczności użycia składowych harmonicznych $e^{+jk_1\cdot\frac{2\pi}{N_1}\cdot n_1}$ oraz $e^{+jk_2\cdot\frac{2\pi}{N_2}\cdot n_2}$. Odpowiednia suma tych składowych (patrz wzór 2.1) pozwoli prawidłowo „zbudować” treść obrazu.

$$f(n_1,n_2) = \frac{1}{\sqrt{N_1\cdot N_2}}\cdot\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1} F(k_1,k_2)\cdot e^{+jk_1\cdot\frac{2\pi}{N_1}\cdot n_1}\cdot e^{+jk_2\cdot\frac{2\pi}{N_2}\cdot n_2} =$$

$$= \frac{1}{\sqrt{N_1\cdot N_2}}\cdot\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1} F(k_1,k_2)\cdot e^{\left(+jk_1\cdot\frac{2\pi}{N_1}\cdot n_1+jk_2\cdot\frac{2\pi}{N_2}\cdot n_2\right)} \qquad (2.1)$$

Ale co to znaczy odpowiednia suma składowych harmonicznych? Żeby w drodze powyższej sumy dało się odtworzyć treść obrazu $f(n_1,n_2)$, każda ze składowych harmonicznych (czyli o określonych częstotliwościach zmian wartości funkcji) musi zostać pomnożona przez ściśle określoną wartość $F(k_1,k_2)$. Wartość ta jest obliczana z poniższego wzoru, który jest w literaturze znany jako **dwuwymiarowe dyskretne przekształcenie Fouriera** (ang. two-dimensional Discrete Fourier Transformation – 2D-DFT).

$$F(k_1,k_2)=\frac{1}{\sqrt{N_1\cdot N_2}}\cdot\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}f(n_1,n_2)\cdot e^{-jk_1\cdot\frac{2\pi}{N_1}\cdot n_1}\cdot e^{-jk_2\cdot\frac{2\pi}{N_2}\cdot n_2}=$$

$$=\frac{1}{\sqrt{N_1\cdot N_2}}\cdot\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}f(n_1,n_2)\cdot e^{\left(-jk_1\cdot\frac{2\pi}{N_1}\cdot n_1-jk_2\cdot\frac{2\pi}{N_2}\cdot n_2\right)} \tag{2.2}$$

Wartości $F(k_1,k_2)$ są w ogólności liczbami zespolonymi. Mogą się więc składać z części rzeczywistej $Re\{F(k_1,k_2)\}$ oraz części urojonej $Im\{F(k_1,k_2)\}$, co matematycznie przedstawiamy w następujący sposób:

$$F(k_1,k_2)=Re\{F(k_1,k_2)\}+j\cdot Im\{F(k_1,k_2)\} \tag{2.3}$$

gdzie j jest jednostką urojoną.

Każdą liczbę zespoloną można z kolei przedstawić w postaci wykładniczej, jak w wyrażeniu poniżej:

$$F(k_1,k_2)=|F(k_1,k_2)|\cdot e^{j\varphi(k_1,k_2)} \tag{2.4}$$

gdzie $|F(k_1,k_2)|$ jest modułem liczby zespolonej $F(k_1,k_2)$, natomiast $\varphi(k_1,k_2)=arg\{F(k_1,k_2)\}$ jest argumentem tej liczby.

Podstawiając wykładniczą postać funkcji $F(k_1,k_2)$ (wzór 2.4) do wyrażenia 2.1 otrzymujemy następującą formułę:

$$f(n_1,n_2)=\frac{1}{\sqrt{N_1\cdot N_2}}\cdot\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1}|F(k_1,k_2)|\cdot e^{j\varphi(k_1,k_2)}\cdot e^{\left(+jk_1\cdot\frac{2\pi}{N_1}\cdot n_1+jk_2\cdot\frac{2\pi}{N_2}\cdot n_2\right)}=$$

$$=\frac{1}{\sqrt{N_1\cdot N_2}}\cdot\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1}|F(k_1,k_2)|\cdot e^{\left(+jk_1\cdot\frac{2\pi}{N_1}\cdot n_1+jk_2\cdot\frac{2\pi}{N_2}\cdot n_2+j\varphi(k_1,k_2)\right)}= \tag{2.5}$$

$$=\frac{1}{\sqrt{N_1\cdot N_2}}\cdot\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1}|F(k_1,k_2)|\cdot e^{+j\left(k_1\cdot\frac{2\pi}{N_1}\cdot n_1+k_2\cdot\frac{2\pi}{N_2}\cdot n_2+\varphi(k_1,k_2)\right)}$$

Żeby suma zespolonych **składowych** harmonicznych **dała w wyniku obraz** $f(n_1,n_2)$, poszczególne **składowe** (czyli o określonej wartości częstotliwości) **muszą mieć ściśle określoną amplitudę** (równą $|F(k_1,k_2)|$) **oraz konkretną wartość fazy początkowej** (która wynosi $\varphi(k_1,k_2)$). Ten ważny wniosek wynika bezpośrednio z matematycznej postaci wyrażenia 2.5. Z uwagi na przedstawione znaczenie wielkości $|F(k_1,k_2)|$ oraz $\varphi(k_1,k_2)$ są one w teorii sygnałów nazywane odpowiednio **widmem amplitudowym** oraz **widmem fazowym** sygnału $f(n_1,n_2)$.

W powyższym wyrażeniu z funkcji zespolonej składowej harmonicznej można wydzielić część rzeczywistą oraz część urojoną, stosując wzór Eulera dla tej funkcji:

$$e^{+j\left(k_1\cdot\frac{2\pi}{N_1}\cdot n_1+k_2\cdot\frac{2\pi}{N_2}\cdot n_2+\varphi(k_1,k_2)\right)} = cos\left(k_1\cdot\frac{2\pi}{N_1}\cdot n_1+k_2\cdot\frac{2\pi}{N_2}\cdot n_2+\varphi(k_1,k_2)\right)+ j\cdot sin\left(k_1\cdot\frac{2\pi}{N_1}\cdot n_1+k_2\cdot\frac{2\pi}{N_2}\cdot n_2+\varphi(k_1,k_2)\right) \quad (2.6)$$

Uwzględnienie otrzymanego wyniku we wzorze 2.5 prowadzi do następującego wyrażenia:

$$f(n_1,n_2)=\frac{1}{\sqrt{N_1\cdot N_2}}\cdot\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1}|F(k_1,k_2)|\cdot\left\{\begin{matrix} cos\left(k_1\cdot\frac{2\pi}{N_1}\cdot n_1+k_2\cdot\frac{2\pi}{N_2}\cdot n_2+\varphi(k_1,k_2)\right)+ \\ +j\cdot sin\left(k_1\cdot\frac{2\pi}{N_1}\cdot n_1+k_2\cdot\frac{2\pi}{N_2}\cdot n_2+\varphi(k_1,k_2)\right)\end{matrix}\right\} \quad (2.7)$$

Z uwagi na obecność we wzorze 2.7 funkcji kosinusowych i sinusowych, które reprezentują odpowiednio część rzeczywistą oraz część urojoną funkcji $f(n_1,n_2)$, powyższa formuła pozwala na matematyczny opis nawet takich obrazów, których próbki są wartościami zespolonymi. W przypadku obrazów o rzeczywistych wartościach próbek (a tak jest w przypadku znanych nam obrazów jednej składowej) obraz daje się reprezentować przy pomocy samych składowych kosinusoidalnych bądź samych składowych sinusoidalnych, jeśli dopuścimy możliwość niezerowej wartości fazy początkowej składowym harmonicznym. W każdym z wymienionych dwóch alternatywnych przypadków potrzebujemy więc do tego celu **przebiegów kosinus, bądź alternatywnie, sinus o ściśle określonej amplitudzie, częstotliwości oraz fazie początkowej**. To, która z tych dwóch funkcji lepiej spełni swoją rolę w zadaniach kompresji obrazów scen naturalnych będzie przedmiotem szerszej dyskusji w dalszej części książki.

2.5. Dyskretne przekształcenie Fouriera – dalszy komentarz

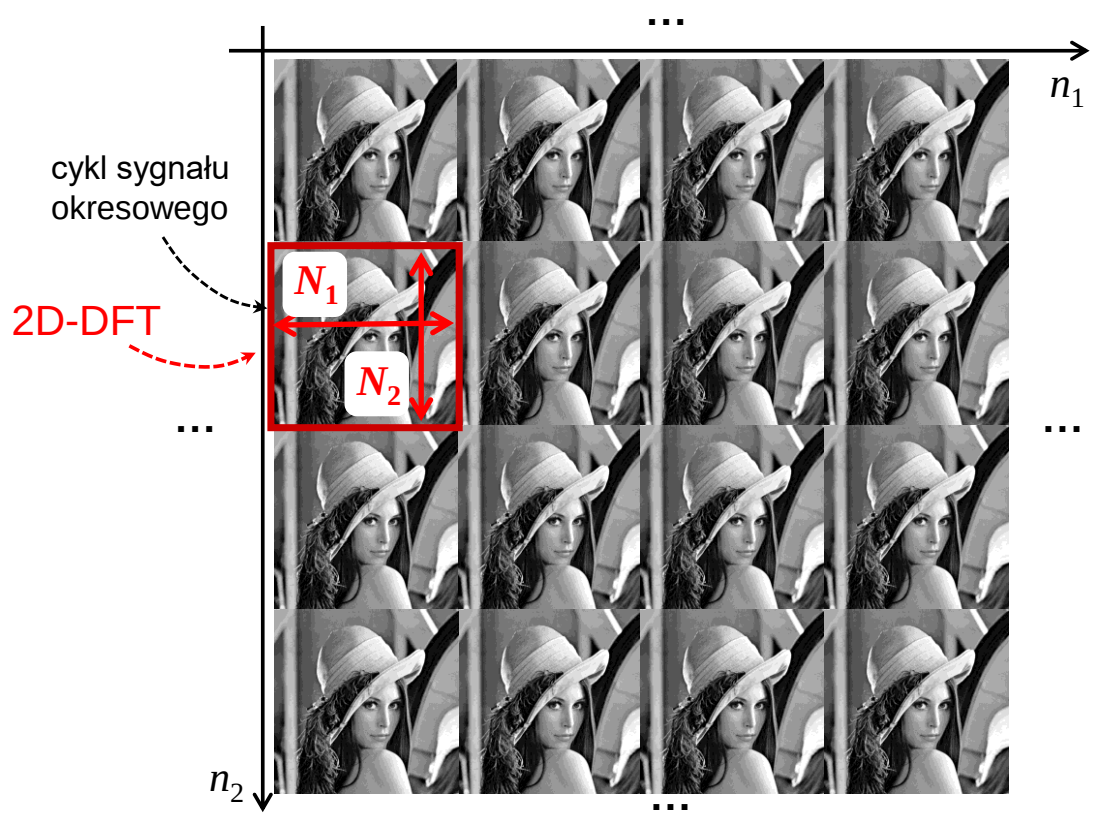
Żeby idea „częstotliwościowej" reprezentacji obrazu mogła być użyta w praktyce, to stosowany do przedstawienia obrazu zbiór składowych harmonicznych nie może być zbiorem nieskończonym. Z tej perspektywy konieczne jest zastosowanie skończonego zbioru składowych harmonicznych, których częstotliwości są z góry ściśle określone.

Z teorii sygnałów wiadomo, że powyższe wymagania będą spełnione tylko w jednym przypadku – jeśli sygnał $f(n_1,n_2)$ będzie dyskretny (czy cyfrowy) oraz dodatkowo okresowy[17]. W tym konkretnym przypadku sygnał uda się przedstawić przy pomocy skończonego zbioru składowych harmonicznych, których częstotliwości będą całkowitą (a nie dowolną rzeczywistą) wielokrotnością częstotliwości $\frac{1}{N_1}$ tzw. podstawowej (czyli pierwszej) składowej harmonicznej poziomej oraz całkowitą wielokrotnością częstotliwości $\frac{1}{N_2}$ tzw. podstawowej (czyli pierwszej) składowej harmonicznej pionowej. W przedstawionych w poprzednim punkcie formułach matematycznych, skalowanie częstotliwości podstawowej składowej harmonicznej jest realizowane przez indeksy k_1 oraz k_2, odpowiednio dla składowej harmonicznej poziomej oraz pionowej. Zatem tylko „częstotliwościowa" analiza **dyskretnych sygnałów okresowych** może być

[17] Zarówno w przypadku sygnałów ciągłych (okresowe i nieokresowe), jak również dyskretnych sygnałów nieokresowych potrzebujemy nieskończonej liczby składowych harmonicznych do reprezentacji tych sygnałów.

przedmiotem pełnego jej wdrożenia w praktyce kompresji obrazów. Ponieważ liczba otrzymywanych składowych harmonicznych nie będzie w takim przypadku nieskończona.

Dyskretne przekształcenie Fouriera (opisane w poprzednim punkcie) jest właśnie zdefiniowane dla **dyskretnych sygnałów okresowych**. Z punktu widzenia więc tego przekształcenia obraz, o wymiarach $N_1 \times N_2$, na którym to przekształcenie jest liczone jest jednym cyklem okresowego sygnału, tak jak zostało to zilustrowane na poniższym rysunku.



Rysunek 2-14 Postać okresowa obrazu „Lena” (patrz rysunek 2-8), która jest widziana z perspektywy dyskretnego przekształcenia Fouriera. Okresy sygnału wynoszą N_1 oraz N_2, odpowiednio w kierunkach poziomym oraz pionowym w sygnale.

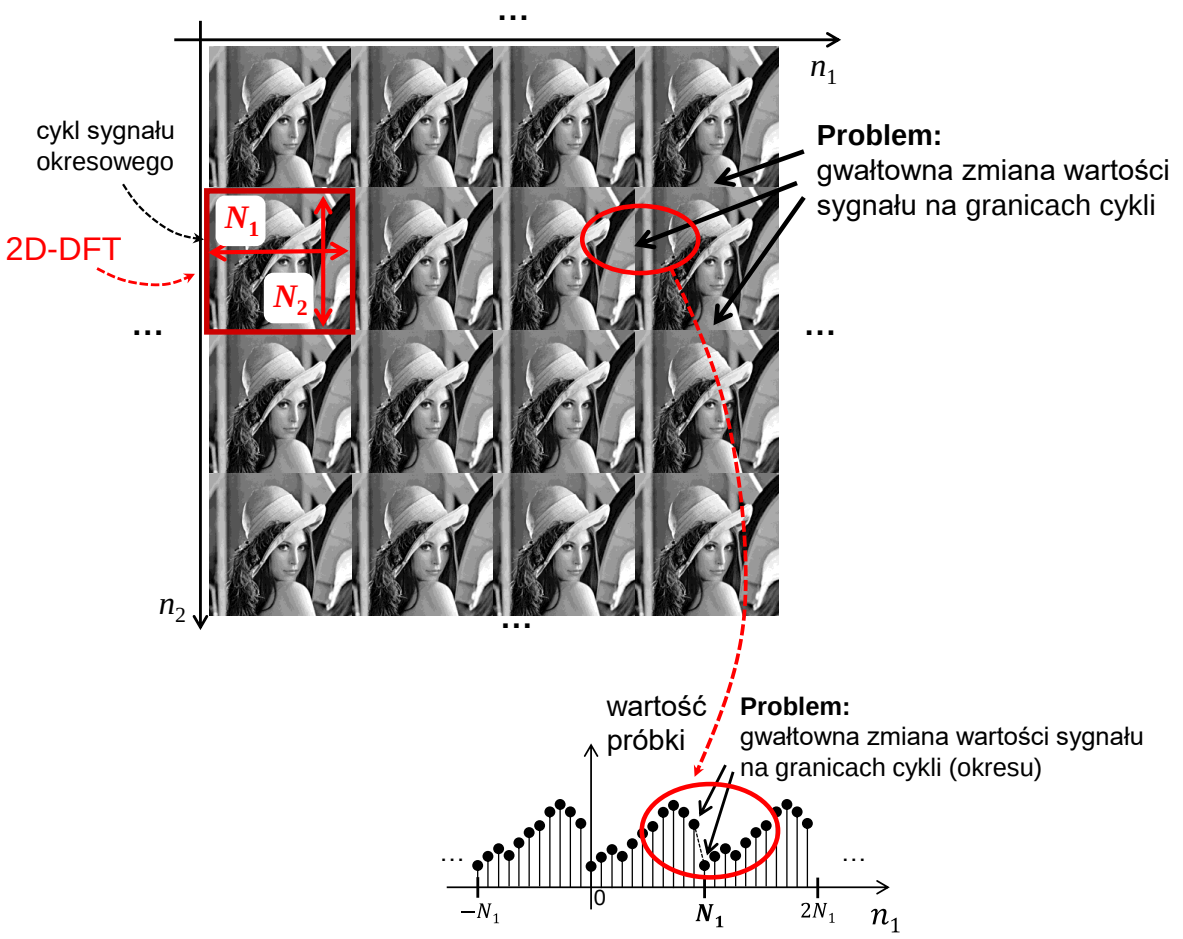
Z perspektywy przedstawionych wcześniej formuł matematycznych możemy powiedzieć, że dyskretne przekształcenie Fouriera przyporządkowuje zbiorowi $N_1 \times N_2$ próbek sygnału, które reprezentują jeden cykl okresowego obrazu w dziedzinie przestrzennej, zbiór $N_1 \times N_2$ próbek widma, które również jest okresowe[18]. W próbkach tego widma „ukryta” jest informacja o amplitudzie oraz fazie początkowej zespolonych składowych harmonicznych, które posłużą do odtworzenia treści obrazu.

[18] Okresowość widma wynika z cech reprezentacji „częstotliwościowej” sygnału dyskretnego – widmo takiego sygnału jest zawsze okresowe. Jednak w ramach jednego cyklu widma okresowego mamy tutaj skończoną liczbę składowych harmonicznych.

2.6. Praktyczne użycie dyskretnego przekształcenia Fouriera – istotny problem

Za pomocą dyskretnego przekształcenia Fouriera dokonujemy „częstotliwościowej” analizy, a później reprezentacji sygnału okresowego. W sygnale tym, na granicach każdego cyklu może dochodzić do gwałtownej zmiany wartości sygnału. Można to zaobserwować na rysunku 2-15. Z punktu widzenia reprezentacji częstotliwościowej i później kompresji obrazu, ta skokowa zmiana wartości sygnału jest istotnym problemem. Odwzorowanie tego skoku w dziedzinie częstotliwości wymagać będzie użycia wysokoczęstotliwościowych składowych, ponieważ same składowe o częstotliwościach niskich nie „oddadzą” tej szybkiej zmiany wartości sygnału. Ale na tym problem się nie kończy. Wspomniane wysokoczęstotliwościowe składowe posiadają niezerową wartość nie tylko w miejscu wystąpienia skoku w okresowym sygnale, ale również w miejscach gdzie mamy już treść naszego obrazu. W tych właśnie miejscach, żeby skompensować istnienie niezerowych próbek składowych wysokoczęstotliwościowych (których obecność, podkreślmy raz jeszcze, jest wynikiem skoku a nie treści samego obrazu), inne wartości przyjąć muszą pozostałe próbki widma (albo co najmniej część tych próbek). To ostatnie, jeśli pominie się kwestię kosztu bitowego reprezentacji próbek widma, nie jest problemem w przypadku realizacji kodowania bezstratnego obrazu. Jednak w przypadku kompresji stratnej może to być już jakimś problemem, ponieważ późniejsza eliminacja w procesie kodowania obrazu składowych wysokoczęstotliwościowych (co jest powszechną praktyką w kompresji stratnej) spowoduje, że te zmienione na skutek skoku pozostałe próbki widma nie mają już czego kompensować i owa zmiana wartości próbek widma objawi się określonym zniekształceniem w obrazie, który zostanie odtworzony w dekoderze.

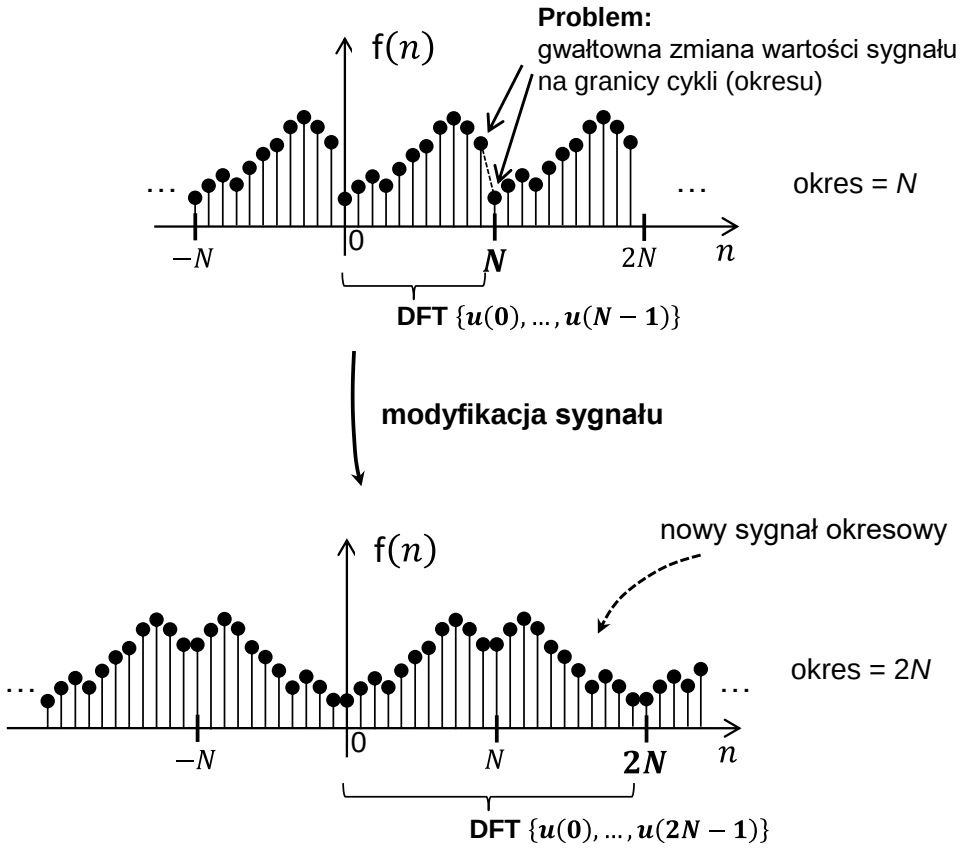
Występująca na granicach cykli okresowego sygnału gwałtowna zmiana jego wartości prowadzi zatem do pewnego skomplikowania opisu sygnału w dziedzinie częstotliwości, co w praktyce przekłada się na zwiększony koszt bitowy reprezentacji obrazu. Jest to oczywiście niepożądane, ponieważ te dodatkowe bity służą opisowi „zjawiska” w sygnale, które nie jest elementem treści cyklu sygnału okresowego (czyli treści samego obrazu), tylko jest opisem tego co zachodzi w sygnale okresowym na granicach jego cykli.



Rysunek 2-15 Ilustracja problemu „skokowej” zmiany wartości sygnału okresowego na granicach każdego cyklu. W wyniku tego skoku reprezentacja częstotliwościowa obrazu staje się bardziej złożona, co przekłada się na zwiększony bitowy koszt opisu wszystkich składowych.

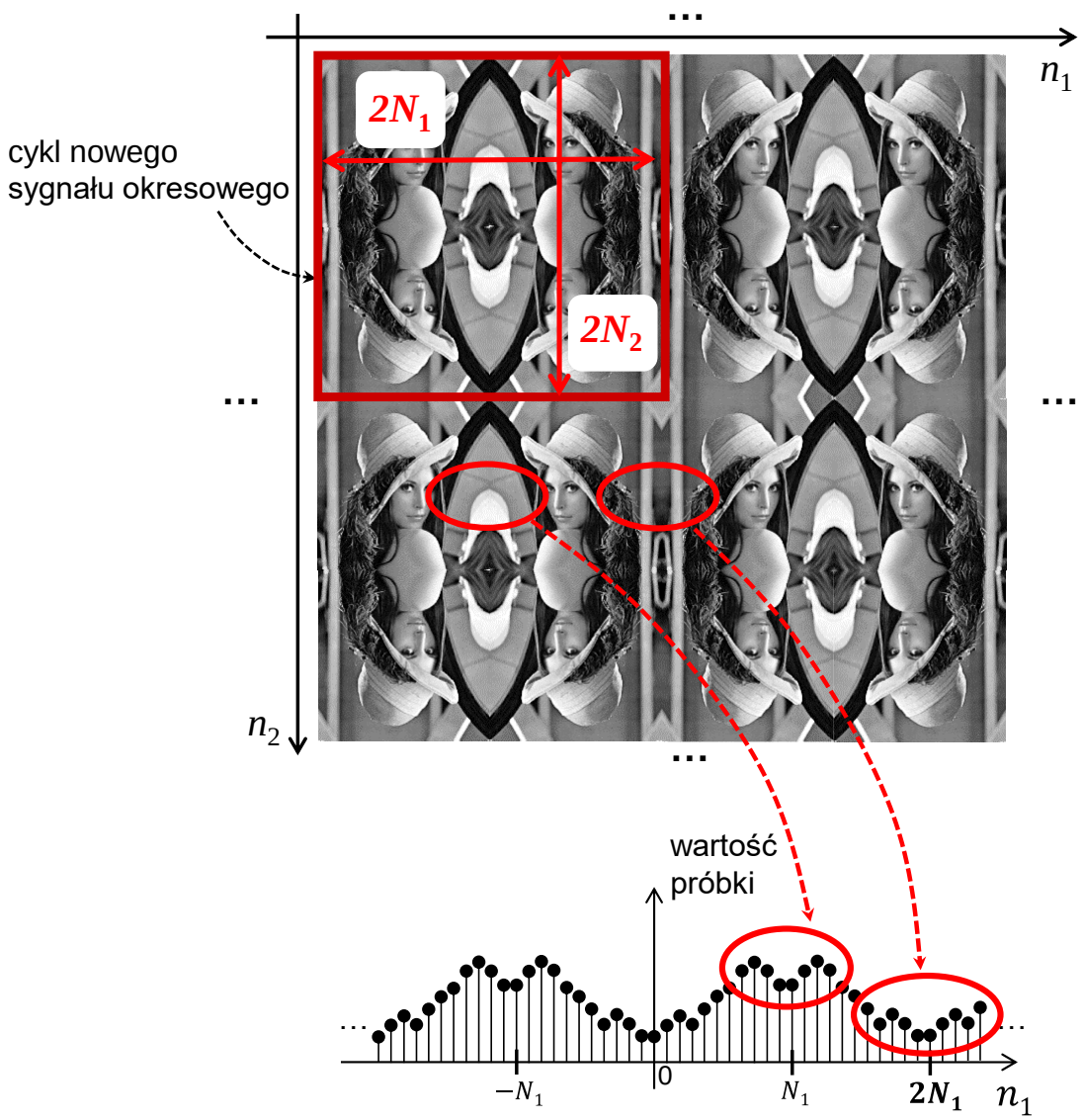
2.7. Eliminacja problemu „skokowej” zmiany wartości sygnału na granicach cykli

Rozwiązaniem przedstawionego w poprzednim punkcie problemu jest odpowiednia modyfikacja pierwotnego sygnału okresowego. W pierwotnym sygnale, który zawiera „skoki” wystarczy lustrzenie odbić co drugi cykl sygnału (względem pionowej osi, która przebiega przez środek cyklu), a wtedy powstanie nowy sygnał okresowy, którego postać została przedstawiona na poniższym rysunku.



Rysunek 2-16 Ilustracja sposobu rozwiązania problemu „skokowej” zmiany wartości sygnału okresowego na granicach cykli. W efekcie tego otrzymywany się nowy sygnał okresowy, którego okres jest dwukrotnie dłuższy w stosunku do okresu sygnału pierwotnego.

W przypadku obrazu, który jest sygnałem dwuwymiarowym, wspomnianą modyfikację należy wykonać w obu kierunkach obrazu: poziomym i pionowym. W wyniku tej operacji otrzymywany jest nowy sygnał okresowy (patrz rysunek 2-17), którego okres jest dwukrotnie dłuższy w porównaniu z okresem sygnału przed modyfikacją. **Brak „skokowej” zmiany wartości** nowego sygnału na granicach kolejnych cykli, jest jego cechą charakterystyczną i **upraszcza opis tego sygnału w dziedzinie częstotliwości**. Ma to szczególnie duże znaczenie z punktu widzenia zastosowania metod analizy częstotliwościowej sygnału w zadaniach jego kompresji. Z tej perspektywy dyskretne przekształcenie Fouriera (DFT) lepiej jest liczyć na zmienionej wersji sygnału okresowego. Kolejny punkt odpowie na pytanie, co jest wynikiem takiego działania.



Rysunek 2-17 Odbicie lustrzane co drugiego cyklu sygnału z rysunku 2-15 prowadzi do nowego sygnału okresowego, w którym nie ma już skokowych zmian wartości sygnału na granicach cykli.

2.8. DFT „nowego sygnału okresowego” – czyli w efekcie dyskretne przekształcenie kosinusowe (DCT)

2.8.1. DCT sygnału jednowymiarowego, czyli 1D-DCT

2.8.1.1. Wyprowadzenie matematyczne

Zastosowanie przekształcenia DFT na nowym sygnale okresowym prowadzi do bardzo ważnego, z punktu widzenia praktyki kompresji obrazów, rezultatu. Tym rezultatem jest nowe przekształcenie, które jest w literaturze dobrze znane jako **dyskretne przekształcenie kosinusowe** (ang. Discrete Cosine Transformation – DCT) [Ahmed74]. Żeby w pełniejszy sposób zrozumieć istotę nowego przekształcenia, konieczne będzie zapoznanie się z jego matematycznym wyprowadzeniem. W celu uproszczenia, przedstawiony w tym punkcie opis matematyczny będzie

dotyczył jednowymiarowego sygnału (czyli funkcji jednej zmiennej), jednak wnioski, które tutaj otrzymamy będą również prawdziwe dla dwuwymiarowych sygnałów, czyli obrazów.

Punktem startowym naszego wyprowadzenia jest przekształcenie DFT, liczone na nowym sygnale okresowym (patrz rysunek 2-16). Sygnał ten posiada okres $2N$, zatem uwzględnienie tego faktu we wzorze na współczynniki DFT jednowymiarowego sygnału[19] prowadzi do następującego równania:

$$F(k) = \frac{1}{\sqrt{2N}} \cdot \sum_{n=0}^{2N-1} f(n) \cdot e^{-jk \cdot \frac{2\pi}{2N} \cdot n} = \\ = \frac{1}{\sqrt{2N}} \cdot \left[\sum_{n=0}^{N-1} f(n) \cdot e^{-jk \cdot \frac{2\pi}{2N} \cdot n} + \sum_{n=N}^{2N-1} f(n) \cdot e^{-jk \cdot \frac{2\pi}{2N} \cdot n} \right] \tag{2.8}$$

Nowy sygnał okresowy posiada symetrię, którą można wyrazić równaniem:

$$f(n) = f(2N-1-n), \quad \text{dla } n = 0, 1, \dots, N-1 \tag{2.9}$$

Uwzględniając powyższy związek możemy nieco inaczej wyrazić drugą sumę, która występuje we wzorze na $F(k)$ (patrz równanie 2.8):

$$\sum_{n=N}^{2N-1} f(n) \cdot e^{-jk \cdot \frac{2\pi}{2N} \cdot n} = \sum_{n=0}^{N-1} f(n) \cdot e^{-jk \cdot \frac{2\pi}{2N} \cdot (2N-1-n)}$$

Uwzględnienie tego faktu we wzorze na DFT nowego sygnału prowadzi do następującej zależności:

$$F(k) = \frac{1}{\sqrt{2N}} \cdot \left[\sum_{n=0}^{N-1} f(n) \cdot e^{-jk \cdot \frac{2\pi}{2N} \cdot n} + \sum_{n=0}^{N-1} f(n) \cdot e^{-jk \cdot \frac{2\pi}{2N} \cdot (2N-1-n)} \right] = \\ = \frac{1}{\sqrt{2N}} \cdot \sum_{n=0}^{N-1} f(n) \cdot \left[e^{-jk \cdot \frac{2\pi}{2N} \cdot n} + e^{-jk \cdot \frac{2\pi}{2N} \cdot (2N-1-n)} \right] \tag{2.10}$$

Drugi składnik powyższego wzoru można rozpisać następująco:

$$e^{-jk \cdot \frac{2\pi}{2N} \cdot (2N-1-n)} = e^{-jk \cdot \frac{2\pi}{2N} \cdot 2N} \cdot e^{+jk \cdot \frac{2\pi}{2N}} \cdot e^{+jk \cdot \frac{2\pi}{2N} \cdot n} = 1 \cdot e^{+jk \cdot \frac{2\pi}{2N}} \cdot e^{+jk \cdot \frac{2\pi}{2N} \cdot n} \tag{2.11}$$

Podstawiając otrzymany wynik do wzoru na $F(k)$ (wzór 2.10) otrzymujemy następującą zależność:

$$F(k) = \frac{1}{\sqrt{2N}} \cdot \sum_{n=0}^{N-1} f(n) \cdot \left[e^{-jk \cdot \frac{2\pi}{2N} \cdot n} + e^{+jk \cdot \frac{2\pi}{2N}} \cdot e^{+jk \cdot \frac{2\pi}{2N} \cdot n} \right] \tag{2.12}$$

[19] W przypadku sygnału okresowego o okresie N współczynniki $F(k)$ przekształcenia DFT wyznacza się ze wzoru:
$F(k) = \frac{1}{\sqrt{N}} \cdot \sum_{n=0}^{N-1} f(n) \cdot e^{-jk \cdot \frac{2\pi}{N} \cdot n}$

W celu uzyskania zakładanego celu (reprezentacja sygnału przy pomocy samych funkcji kosinusowych!) lewą oraz prawą stronę ostatniego równania mnożymy przez czynnik $e^{-jk\cdot\frac{\pi}{2N}}$, co prowadzi do następującego równania:

$$e^{-jk\cdot\frac{\pi}{2N}}\cdot F(k) = e^{-jk\cdot\frac{\pi}{2N}}\cdot\frac{1}{\sqrt{2N}}\cdot\sum_{n=0}^{N-1} f(n)\cdot\left[e^{-jk\cdot\frac{2\pi}{2N}\cdot n} + e^{+jk\cdot\frac{2\pi}{2N}}\cdot e^{+jk\cdot\frac{2\pi}{2N}\cdot n}\right] =$$

$$= \frac{1}{\sqrt{2N}}\cdot\sum_{n=0}^{N-1} f(n)\cdot\left[e^{-jk\cdot\frac{2\pi}{2N}\cdot n}\cdot e^{-jk\cdot\frac{\pi}{2N}} + e^{-jk\cdot\frac{\pi}{2N}}\cdot e^{+jk\cdot\frac{2\pi}{2N}}\cdot e^{+jk\cdot\frac{2\pi}{2N}\cdot n}\right] = \tag{2.13}$$

$$= \frac{1}{\sqrt{2N}}\cdot\sum_{n=0}^{N-1} f(n)\cdot\left[e^{-jk\cdot\frac{2\pi}{2N}\cdot n}\cdot e^{-jk\cdot\frac{\pi}{2N}} + e^{+jk\cdot\frac{\pi}{2N}}\cdot e^{+jk\cdot\frac{2\pi}{2N}\cdot n}\right]$$

Występujące w kwadratowym nawiasie funkcje $e^{j\dots}$ można zapisać w bardziej kompaktowej formie:

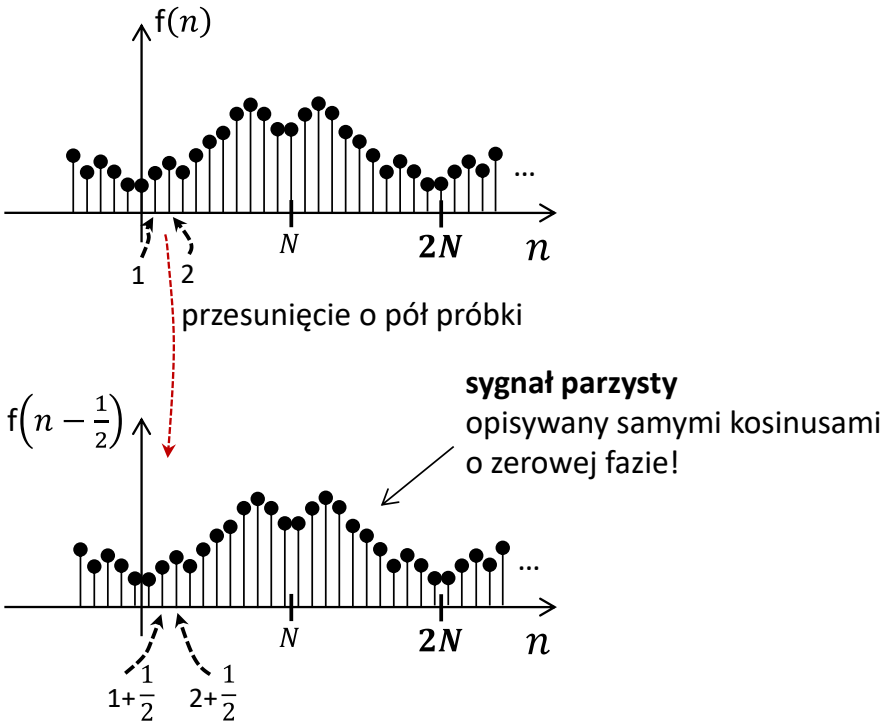
$$e^{-jk\cdot\frac{\pi}{2N}}\cdot F(k) = \frac{1}{\sqrt{2N}}\cdot\sum_{n=0}^{N-1} f(n)\cdot\left[e^{-jk\cdot\frac{\pi}{2N}\cdot(2n+1)} + e^{+jk\cdot\frac{\pi}{2N}\cdot(2n+1)}\right] \tag{2.14}$$

Ostatnim już etapem wyprowadzenia jest uwzględnienie wzoru Eulera dla funkcji **kosinus**, w wyniku czego otrzymujemy wzór na nowe przekształcenie:

$$e^{-jk\cdot\frac{\pi}{2N}}\cdot F(k) = \frac{1}{\sqrt{2N}}\cdot\sum_{n=0}^{N-1} f(n)\cdot 2\cdot cos\left(k\cdot\frac{\pi}{2N}\cdot(2n+1)\right), \text{dla } k = 0,\dots,N-1 \tag{2.15}$$

Z uwagi na występującą w tym wzorze funkcję **kosinus** jest ono nazywane **dyskretnym przekształceniem kosinusowym** (ang. Discrete Cosine Transformation – DCT). Obecność pojedynczego wyrażenie na funkcję kosinus świadczy o **jednowymiarowym dyskretnym przekształceniu kosinusowym** (ang. one-dimensional Discrete Cosine Transformation – 1D-DCT). Przedmiotowy sygnał będzie więc reprezentowany jako ważona suma samych tylko funkcji kosinus o określonych częstotliwościach (patrz wzór 2.17 na odwrotne przekształcenie kosinusowe), co będzie jeszcze tematem szerszej dyskusji w punkcie 2.8.3.

W otrzymanym wzorze występuje jeszcze dodany wcześniej składnik $e^{-jk\cdot\frac{\pi}{2N}}$. Z teorii sygnałów wiadomo jednak, że jeśli sygnał okresowy $f(n)$ (o okresie $2N$) posiada próbki widma $F(k)$ to $e^{-jk\cdot\frac{2\pi}{2N}n_0}\cdot F(k)$ będą próbkami widma sygnału $f(n-n_0)$, czyli sygnału $f(n)$ przesuniętego w prawo o n_0 próbek. Przesunięcie sygnału wpływa tylko na zmianę widma fazowego sygnału (czyli zmienia się informacja o fazie początkowej składowych harmonicznych), pozostawiając widmo amplitudowe niezmienionym (czyli informacja o częstotliwościach składowych harmonicznych, przy pomocy których budujemy sygnał oraz informacja o amplitudzie tych składowych nie ulegają zmianie). Obecność składnika $e^{-jk\cdot\frac{\pi}{2N}}$ w równaniu 2.15 sprawia więc, że próbki widma (próbki transformaty kosinusowej) są wyznaczane dla przesuniętej wersji (przesuniętej o pół próbki w prawo, czyli o $\frac{1}{2}$ okresu próbkowania sygnału) sygnału $f(n)$. Ten przesunięty sygnał, w porównaniu z sygnałem oryginalnym, został przedstawiony na rysunku 2-18.



Rysunek 2-18 Sygnał parzysty $f\left(n-\frac{1}{2}\right)$ daje się przedstawić przy pomocy samych tylko funkcji kosinus o zerowej fazie początkowej. Ponieważ sygnał $f(n)$ stanowi przesuniętą wersję parzystego sygnału $f\left(n-\frac{1}{2}\right)$ to reprezentacja sygnału $f(n)$ kosinusami wymaga wprowadzenia dla składowych przesunięcia fazowego, które liniowo zależy od indeksu k. Jednak wartości tych przesunięć są z góry znane.

Znając to przesunięcie możemy je łatwo zniwelować, i z tej perspektywy występujący po lewej stronie wzoru 2.15 składnik $e^{-jk\cdot\frac{\pi}{2N}}$ jest nieistotny i można go pominąć, co prowadzi do wzoru:

$$F(k)=\frac{1}{\sqrt{2N}}\cdot\sum_{n=0}^{N-1}f(n)\cdot 2\cdot cos\left(k\cdot\frac{\pi}{2N}\cdot(2n+1)\right),\text{dla } k=0,\dots,N-1 \qquad (2.16)$$

Odwrotne przekształcenie kosinusowe (ang. Inverse Discrete Cosine Transformation – IDCT) sygnału pozwala odtworzyć (w drodze ważonej sumy funkcji kosinus) próbki sygnału $f(n)$ na podstawie próbek $F(k)$ przekształcenia DCT, zgodnie podanym niżej równaniem:

$$f(n)=\frac{1}{\sqrt{2N}}\cdot\sum_{k=0}^{N-1}F(k)\cdot 2\cdot cos\left(k\cdot\frac{\pi}{2N}\cdot(2n+1)\right),\text{dla } n=0,1,\dots,N-1 \qquad (2.17)$$

Z punktu więc widzenia idei nowego przekształcenia, sygnał jest reprezentowany jako ważona suma składowych kosinusowych o z góry znanym przesunięciu fazowym (fazie początkowej). W przypadku takiego sposobu reprezentacji sygnału do dekodera wysyłalibyśmy tylko dane reprezentujące amplitudę oraz znak składowych kosinusoidalnych o danej częstotliwości, zamiast dużej ilości informacji o wartościach kolejnych próbek sygnału.

Żeby na podstawie samej amplitudy składowych harmonicznych móc ocenić udział każdej ze składowych w mocy całego sygnału (jest to bardzo przydatne z perspektywy kodowania stratnego obrazów, w którym decydujemy o dokładności reprezentacji (bądź całkowitym usunięciu) poszczególnych składowych harmonicznych) konieczne jest wprowadzenie do przedstawionych wcześniej wzorów dodatkowych współczynników skalujących, które oznaczymy jako $C(k)$. Uwzględnienie tego prowadzi do wersji równań (opisujących proste oraz odwrotne przekształcenie DCT), które są przedmiotem praktycznego wykorzystania w kompresji danych multimedialnych.

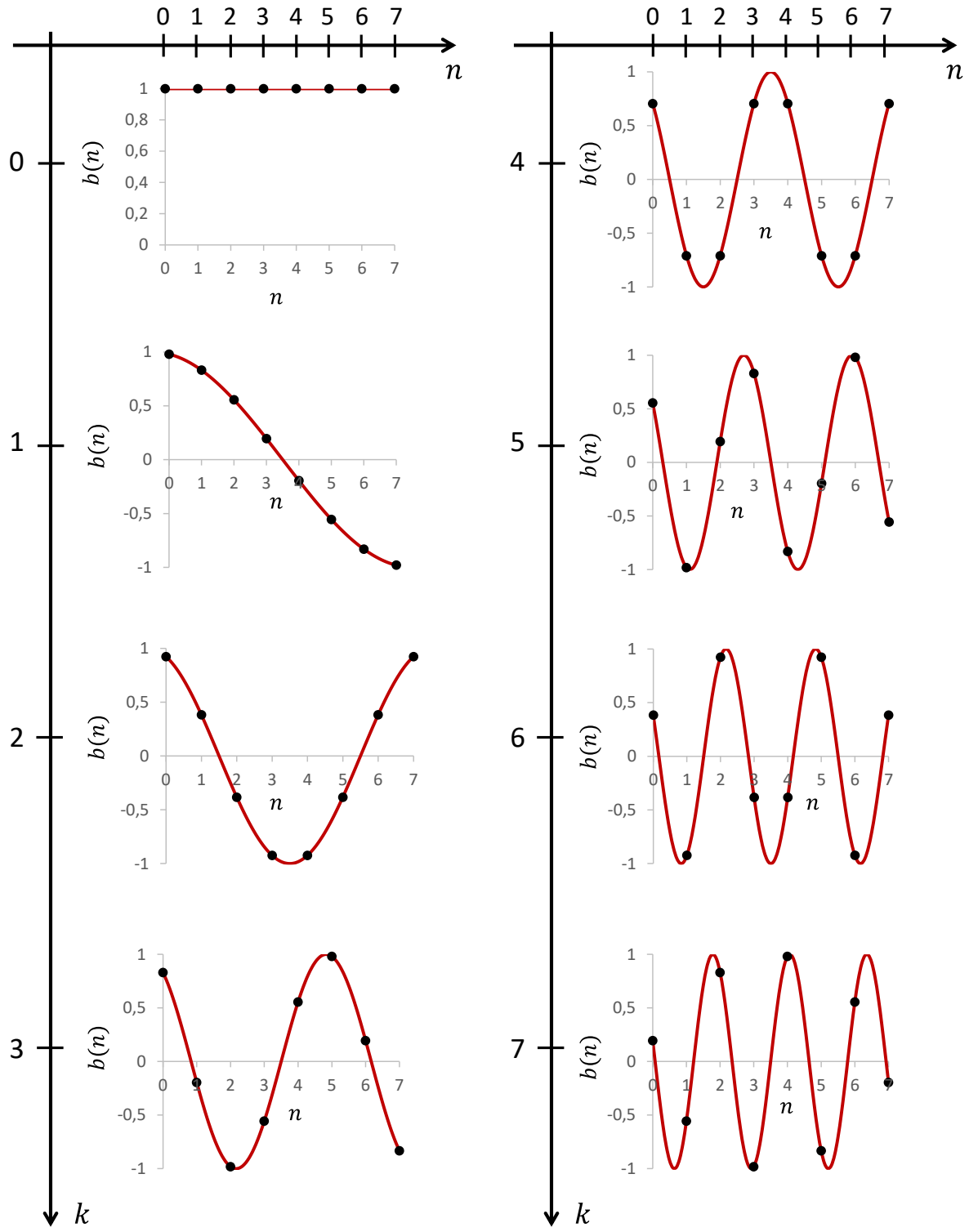
$$F(k) = \frac{1}{\sqrt{2N}} \cdot C(k) \cdot \sum_{n=0}^{N-1} f(n) \cdot 2 \cdot cos\left(k \cdot \frac{\pi}{2N} \cdot (2n+1)\right), \text{dla } k = 0, \dots, N-1 \tag{2.18}$$

$$f(n) = \frac{1}{\sqrt{2N}} \cdot \sum_{k=0}^{N-1} C(k) \cdot F(k) \cdot 2 \cdot cos\left(k \cdot \frac{\pi}{2N} \cdot (2n+1)\right), \text{dla } n = 0, 1, \dots, N-1 \tag{2.19}$$

gdzie $C(k) = \begin{cases} \frac{1}{\sqrt{2}} & \text{dla } k = 0 \\ 1 & \text{dla } k \neq 0 \end{cases}$

2.8.1.2. „Wygląd” kosinusoid przekształcenia 1D-DCT

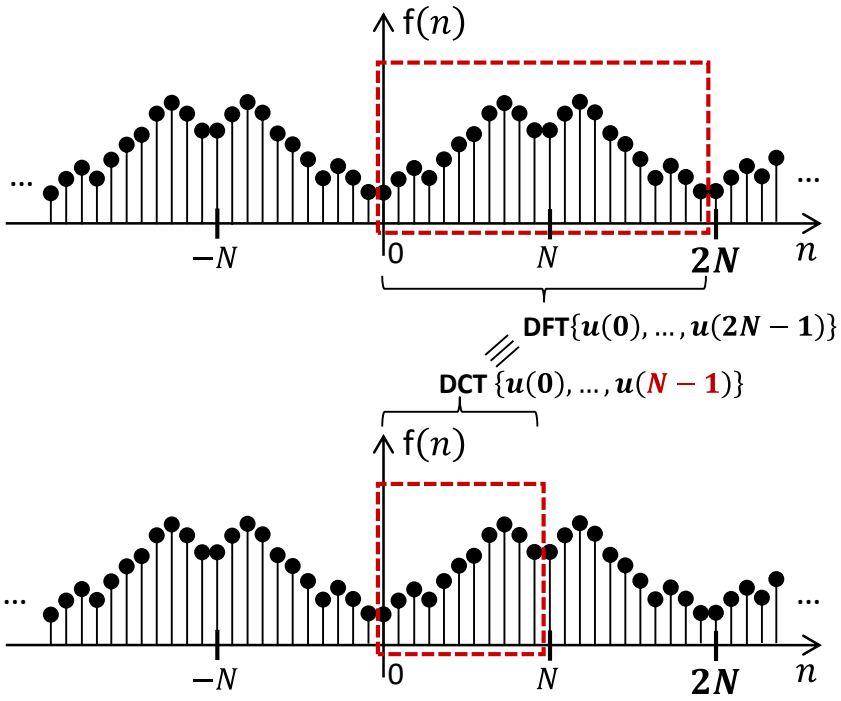
W przypadku przekształcenia 1D-DCT jednowymiarowy sygnał jest przedstawiany jako ważona suma jednowymiarowych kosinusów. Żeby jednak dało się w ten sposób reprezentować dowolny sygnał okresowy (dowolny tzn. wolnozmienny, szybkozmienny, itd.), musimy dysponować „koszykiem” kosinusów, w którym znajdziemy kosinusy o wszystkich możliwych częstotliwościach zmian jego wartości. W przypadku chęci reprezentacji sygnału okresowego, którego jeden cykl zawiera 8 próbek możemy mieć do czynienia z następującymi skrajnościami: wszystkie próbki będą mieć identyczną wartość (czyli częstotliwość zmian wartości próbek będzie zerowa) lub każda z 8 próbek, w stosunku do próbki poprzedniej, będzie zmieniać swoją wartość w sposób gwałtowny (czyli najwyższa częstotliwość zmian wartości próbek). Obsłużenie wszystkich przypadków dla takiego sygnału będzie zatem wymagać „koszyka” z 8 kosinusami (patrz rysunek 2-19), o indeksach częstotliwości k zmieniających się w zakresie od 0 do 7 (z krokiem 1). Ponieważ w przytoczonym przykładzie planujemy reprezentować okresowy sygnał, którego cykl składa się z 8 próbek, to każdy z kosinusów będzie również opisany taką właśnie liczbą próbek (czyli wartości zmiennej n w zakresie od 0 do 7, zmiana z krokiem 1).



Rysunek 2-19 Zestaw **jednowymiarowych kosinusów** będących zbiorem funkcji bazowych przekształcenia 1D-DCT dla $N = 8$. W tym przypadku zestaw składa się z 8 jednowymiarowych kosinusów (o różnych częstotliwościach), przy czym każdy z kosinusów składa się z 8 próbek. W drodze ważonej sumy powyższych kosinusów reprezentowany jest wejściowy sygnał, który zawiera 8 próbek w cyklu. Obrazy kosinusów przygotowano z użyciem własnego oprogramowania. Kosinusy wyznaczono ze wzoru: $b(n) = cos\left(k \cdot \frac{\pi}{2N} \cdot (2n+1)\right)$.

2.8.1.3. Ilustracja związku DCT z DFT

Chociaż związek ten został już przedstawiony z punktu widzenia „języka” matematyki (patrz poprzednie punkty), to zwróćmy raz jeszcze uwagę na graficzną ilustrację tej zależności. Zgodnie z przytoczonym w tym punkcie rysunkiem liczone na $2N$ próbkach nowego sygnału okresowego przekształcenie DFT odpowiada, z punktu widzenia idei, przekształceniu DCT, które jest wyznaczane na bazie N próbek tego sygnału (czyli o połowę mniejszej liczby próbek). Przytoczona zależność jest wynikiem symetrii, jaka występuje w nowym sygnale $f(n)$.



Rysunek 2-20 Ilustracja związku pomiędzy przekształceniami DCT i DFT wyznaczanych dla nowego sygnału okresowego $f(n)$.

2.8.2. DCT sygnału dwuwymiarowego, czyli 2D-DCT

W przypadku obrazu, który jest sygnałem dwuwymiarowym, jesteśmy zainteresowani opisywaniem zmian wartości sygnału, które dokonują się w dwóch kierunkach: poziomym oraz pionowym (a nie w jednym tylko kierunku jak miało to miejsce w przypadku sygnału jednowymiarowego). Dla realizacji tego celu konieczne jest użycie dwóch „paczek” funkcji kosinusowych: funkcji, które „biegną” w kierunku poziomym (wzdłuż zmiennej n_1 w obrazie) i będą reprezentować zmiany treści obrazu dokonujące się właśnie w tym kierunku, oraz funkcji, które umożliwią odwzorowanie pionowych zmian treści (czyli wzdłuż zmiennej n_2 w obrazie). W przytoczonych wcześniej wzorach, opisujących przekształcenie DCT musimy zatem użyć dwóch niezależnych indeksów odpowiadających za częstotliwość: k_1 i k_2, które będą wchodziły w skład argumentu kosinusów odpowiednio „poziomych” oraz „pionowych”, oraz uwzględnić przypadek dwuwymiarowego sygnału $f(n_1, n_2)$. Zastosowanie powyższego prowadzi do **dwuwymiarowego dyskretnego przekształcenia kosinusowego** (ang. two-dimensional discrete cosine transformation – 2D-DCT), które matematycznie jest opisywane parą równań:

$$F(k_1,k_2)=\frac{1}{\sqrt{2N_1}}\cdot\frac{1}{\sqrt{2N_2}}\cdot C(k_1)\cdot C(k_2)\cdot \cdot\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} f(n_1,n_2)\cdot 2\cdot 2\cdot cos\left(k_1\cdot\frac{\pi}{2N_1}\cdot(2n_1+1)\right)\cdot cos\left(k_2\cdot\frac{\pi}{2N_2}\cdot(2n_2+1)\right) \quad (2.20)$$

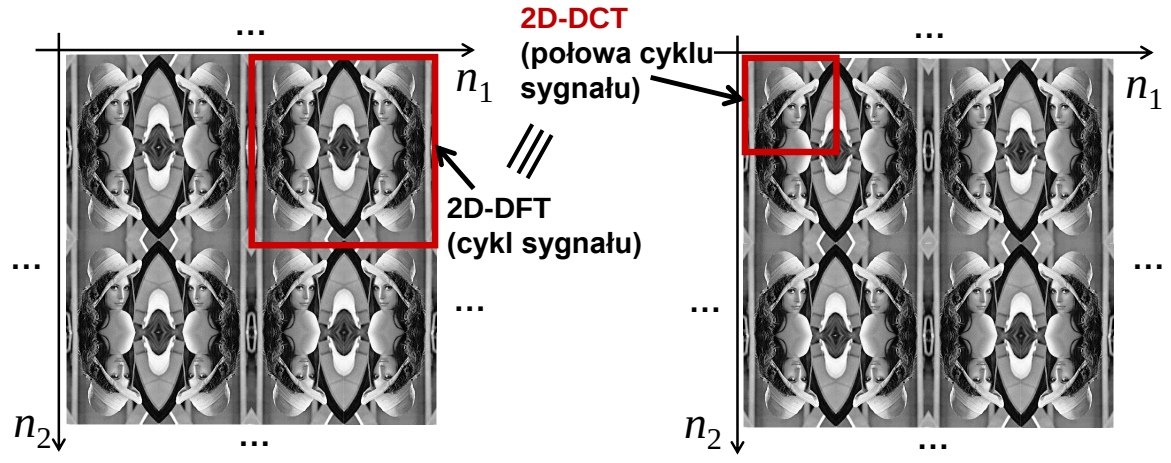
gdzie $0\le k_1\le N_1-1$ oraz $0\le k_2\le N_2-1$.

$$f(n_1,n_2)=\frac{1}{\sqrt{2N_1}}\cdot\frac{1}{\sqrt{2N_2}}\cdot \cdot\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1} C(k_1)\cdot C(k_2)\cdot F(k_1,k_2)\cdot 2\cdot 2\cdot cos\left(k_1\cdot\frac{\pi}{2N_1}\cdot(2n_1+1)\right)\cdot \cdot cos\left(k_2\cdot\frac{\pi}{2N_2}\cdot(2n_2+1)\right) \quad (2.21)$$

gdzie $0\le n_1\le N_1-1$ oraz $0\le n_2\le N_2-1$.

Równania te opisują odpowiednio proste oraz odwrotne dwuwymiarowe przekształcenie DCT. Ponieważ występujący w tych wzorach iloczyn dwóch kosinusów („poziomego” oraz „pionowego”) prowadzi do dwuwymiarowej kosinusoidy (o pulsacjach $k_1\cdot\frac{\pi}{2N_1}\cdot 2$ oraz $k_2\cdot\frac{\pi}{2N_2}\cdot 2$ zmian wartości w kierunkach odpowiednio poziomym oraz pionowym), to w efekcie równanie 2.21 realizuje „budowanie” treści obrazu poprzez ważoną sumę dwuwymiarowych kosinusoid o odpowiednich częstotliwościach.

Pomiędzy przekształceniami 2D-DCT oraz 2D-DFT istnieje analogiczny związek (z punktu widzenia idei tych przekształceń) do tego, który został wcześniej omówiony dla przypadku przekształceń jednowymiarowych. Ilustrację tej zależności prezentuje dodatkowo poniższy rysunek.



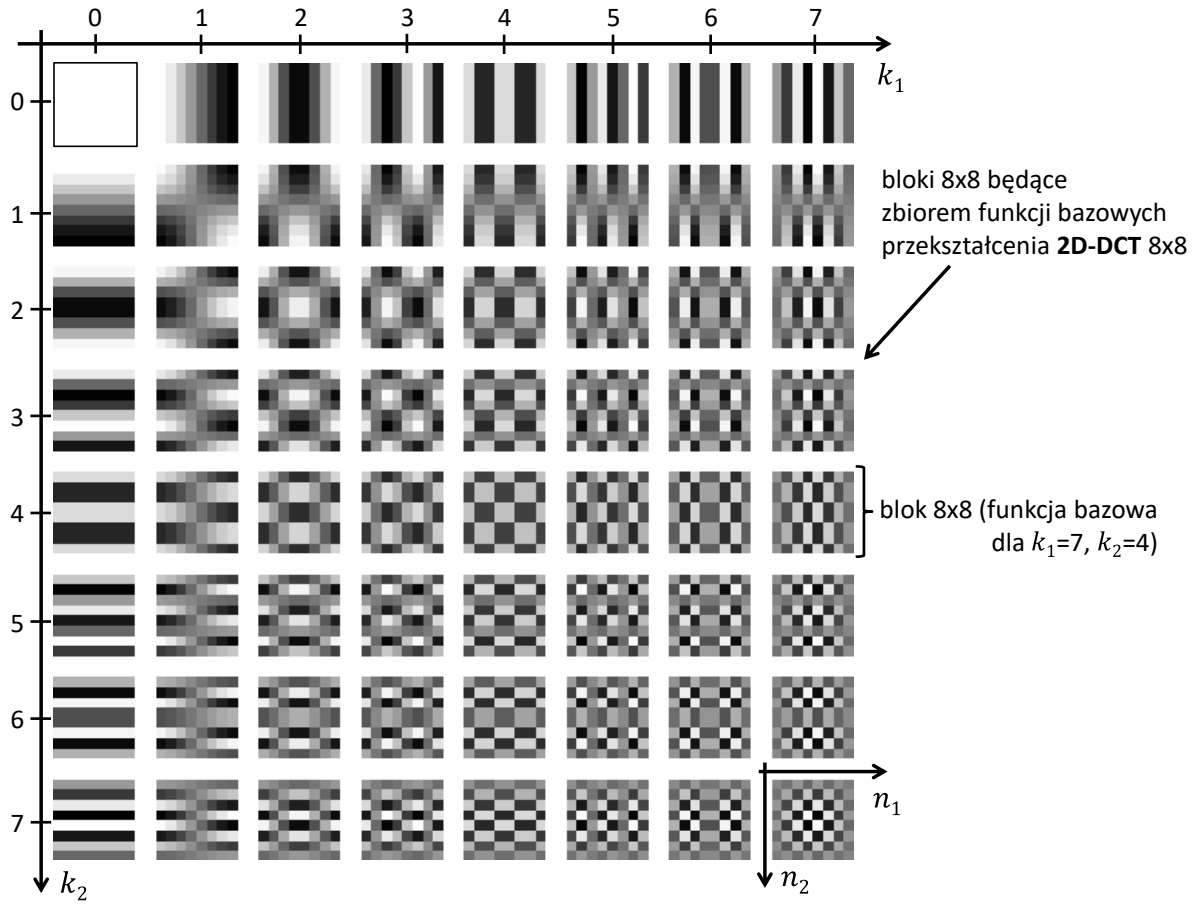
Rysunek 2-21 Ilustracja związku pomiędzy przekształceniami 2D-DCT i 2D-DFT wyznaczanych dla nowego sygnału okresowego $f(n_1,n_2)$. Z punktu widzenia idei 2D-DFT liczone dla cyklu sygnału okresowego sprowadza się do przekształcenia 2D-DCT wyznaczanego dla połowy cyklu tego sygnału.

2.8.3. Dalsza interpretacja oraz znaczenie 2D-DCT w praktyce kompresji obrazów

2.8.3.1. „Wygląd” kosinusoid przekształcenia 2D-DCT

Istota przekształcenia 2D-DCT sprowadza się do budowania treści obrazu za pomocą określonego zestawu dwuwymiarowych kosinusoid. Właśnie takie jest przesłanie matematycznego opisu tego przekształcenia. Żeby lepiej uzmysłowić sobie procedurę tworzenia zestawu **dwuwymiarowych kosinusoid**, oraz zobaczyć jak one wyglądają, przeanalizujmy następujący przykład.

Wyobraźmy sobie obraz o bardzo małym rozmiarze, który składa się z zaledwie 8x8 próbek składowej luminancji. Taki właśnie obraz będziemy chcieli przedstawić poprzez ważoną sumę dwuwymiarowych kosinusoid. Ponieważ treść naszego obrazu może być zupełnie dowolna (może to być jednolite tło, w którym wszystkie próbki mają taką samą wartość, ale również bardzo skomplikowana tekstura, w której wartości próbek zmieniają się z „próbki na próbkę”), to dla realizacji naszego celu potrzebować będziemy takiego zestawu dwuwymiarowych kosinusoid, żeby były one w stanie „oddać” dowolnie szybką zmianę treści obrazu. W zestawie tym muszą się zatem znaleźć kosinusoidy o wszystkich możliwych częstotliwościach przestrzennych, jakie będą mogły wystąpić w obrazie (patrząc w kierunkach poziomym oraz pionowym), przy czym każda z kosinusoid będzie opisana obrazkiem, którego rozmiar jest identyczny z rozmiarem naszego wejściowego obrazu. W analizowanym przez nas przypadku jest to 8x8 próbek. Kolejne wartości próbek każdego z obrazków odzwierciedlają kolejne wartości poszczególnych kosinusów. Po chwili zastanowienia nie trudno jest dojść do wniosku, że reprezentacja obrazów o rozmiarze 8x8 będzie wymagać zestawu 64 kosinusów. Wynika to z tego, że dwa indeksy (k_1 i k_2) określające częstotliwość przestrzenną odpowiednio poziomą i pionową mogą mieć wartość całkowitą z zakresu od 0 do 7 (czyli 8 razy 8 wartości daje w sumie 64). Dla określonej kombinacji wartości k_1 i k_2 wygląd dwuwymiarowych kosinusoid został przedstawiony na rysunku 2-22.



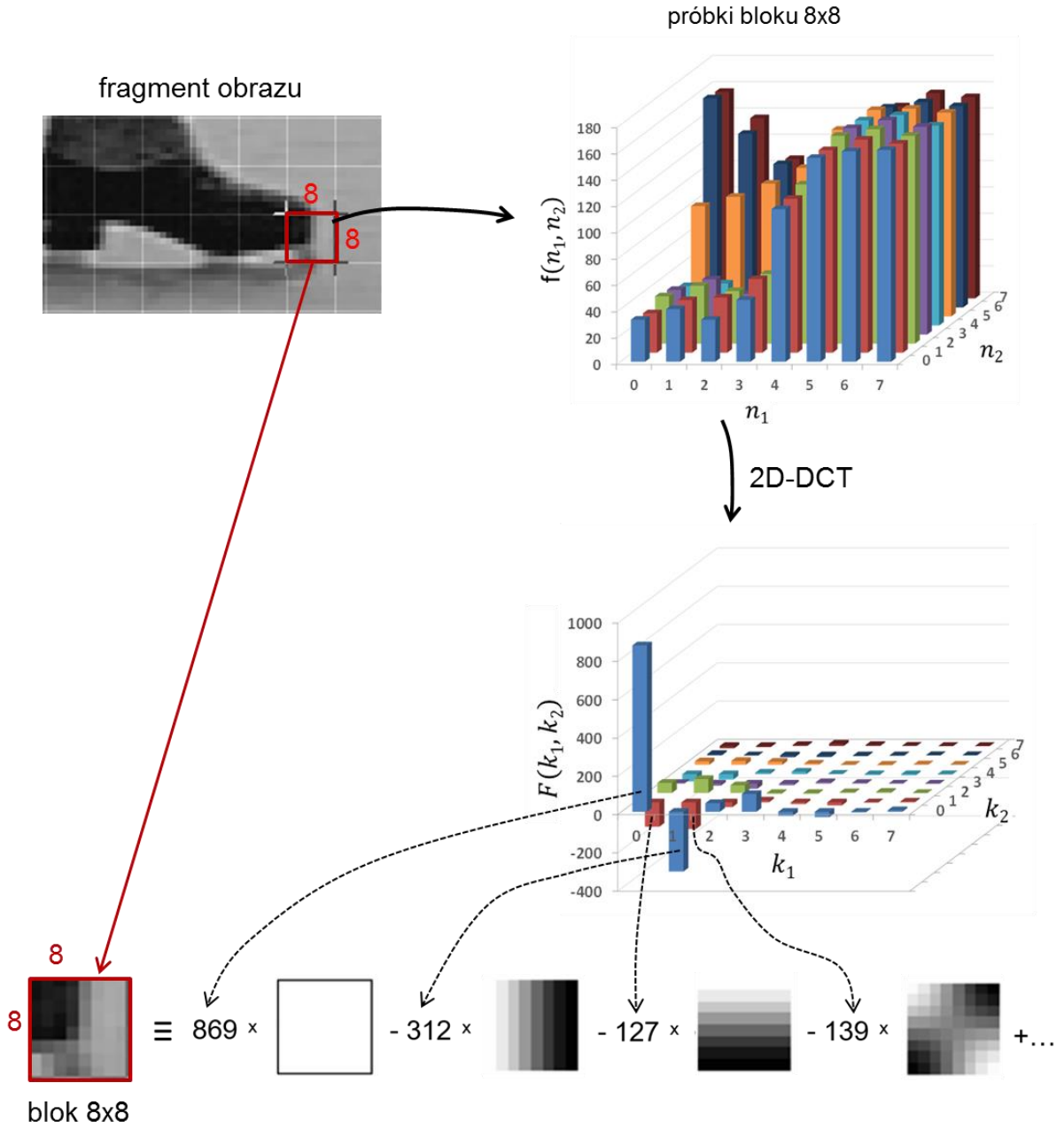
Rysunek 2-22 Zestaw **dwuwymiarowych kosinusoid** będących zbiorem funkcji bazowych przekształcenia 2D-DCT dla $N_1 = 8$, $N_2 = 8$. W tym przypadku zestaw składa się z 64 dwuwymiarowych kosinusoid (o różnych częstotliwościach), przy czym każda z kosinusoid jest obrazkiem o wielkości 8x8 próbek. W drodze ważonej sumy powyższych kosinusów budowana jest treść wejściowego obrazu. Obrazy kosinusów przygotowano z użyciem własnego oprogramowania.

Proszę zauważyć, że istnieje ścisłe powiązanie pomiędzy otrzymanymi obrazkami dwuwymiarowych kosinusoid a postacią jednowymiarowych kosinusów przekształcenia 1D-DCT (patrz punkt 2.8.1.2). Dany kosinus dwuwymiarowy jest tworzony w drodze odpowiedniej „kombinacji” dwóch, ściśle określonych kosinusów jednowymiarowych: jednowymiarowego kosinusa o częstotliwości k_1, który biegnie w kierunku poziomym (czyli wzdłuż zmiennej n_1) oraz jednowymiarowego kosinusa o częstotliwości k_2, który biegnie w kierunku pionowym (czyli wzdłuż zmiennej n_2). Wspomniana „kombinacja” oznacza iloczyn jednowymiarowych kosinusów, czyli w efekcie $\text{kosinus dwuwymiarowy} = cos\left(k_1 \cdot \frac{\pi}{2N_1} \cdot (2n_1 + 1)\right) \cdot cos\left(k_2 \cdot \frac{\pi}{2N_2} \cdot (2n_2 + 1)\right)$.

2.8.3.2. Interpretacja rezultatu 2D-DCT raz jeszcze

Wynikiem liczonego na rzeczywistych próbkach obrazu przekształcenia 2D-DCT są próbki widma, oznaczane jako $F(k_1, k_2)$. Próbki widma są również rzeczywiste, a ich liczba jest taka sama jak liczba próbek obrazu, dla którego wyznaczane jest widmo. Wynikowe wartości $F(k_1, k_2)$ stanowią zatem alternatywny, do wartości $f(n_1, n_2)$, sposób opisu treści obrazu.

Jednak w tym miejscu można sobie zadać pytanie: jaka jest interpretacja otrzymywanych próbek widma? Jak rozumieć wynik przekształcenia kosinusowego? Prostą odpowiedź można sformułować w jednym zdaniu: próbki widma określają amplitudę (równą $|F(k_1, k_2)|$) oraz znak składowych kosinusoidalnych, które będą reprezentować treść obrazu. Suma tych składowych, po ich uprzednim przemnożeniu przez wartości $F(k_1, k_2)$, odwzoruje w dekoderze wartości próbek obrazu, tak jak zostało to przedstawione na poniższym rysunku (patrz też formuła 2.21). Jednak żeby to było możliwe należy wcześniej „odszukać” w obrazie występujących w nim dwuwymiarowych kosinusów, co jest właśnie zadaniem prostego przekształcenia kosinusowego (patrz wzór 2.20).



Rysunek 2-23 Charakter rezultatu przekształcenia 2D-DCT oraz ilustracja znaczenia otrzymanego wyniku.

2.8.3.3. Na czym polega „siła” 2D-DCT?

Dwuwymiarowe przekształcenie kosinusowe znajduje powszechne zastosowanie w koderach obrazu, zarówno statycznego jak i ruchomego, co jest rezultatem następujących cech tego przekształcenia:

1. Jest ono wynikiem zastosowania przekształcenia 2D-DFT dla specjalnie przygotowanej wersji sygnału okresowego, w którym nie ma już skokowych zmian wartości sygnału na granicach cykli (patrz punkt 2.7). Brak tych skokowych zmian wartości prowadzi do uproszczenia widma sygnału w zakresie wysokich częstotliwości. Skoro modyfikując sygnał pozbyliśmy się wspomnianych skoków, to tym samym usunęliśmy z jego reprezentacji częstotliwościowej wysokoczęstotliwościowe składowe harmoniczne, które tę skokową zmianę w sygnale odzwierciedlały. W przypadku stosowania przekształcenia 2D-DCT mamy więc pewność, że ewentualne pojawienie się kosinusów wysokoczęstotliwościowych jest rezultatem odpowiednio złożonej treści obrazu, czyli kosinusy te odzwierciedlają określoną treść obrazu a nie to co się w sygnale dzieje na granicach samych cykli.

2. Dzięki zastosowaniu 2D-DCT (czyli reprezentacji sygnału przy pomocy zdefiniowanego zbioru odpowiednio poprzesuwanych kosinusów) otrzymujemy taką reprezentację częstotliwościową sygnału, w której widmo fazowe sygnału przyjmuje bardzo prostą postać. Otrzymywane widmo fazowe informuje jedynie o tym, jaki jest znak (dodatni lub ujemny) składowej kosinusoidalnej o danej częstotliwości (której amplituda wynosi $|F(k_1, k_2)|$), która jest obecna w naszym sygnale. Prosta postać widma fazowego zmniejsza koszt bitowy reprezentacji widma sygnału.

3. Reprezentując obraz kosinusami możemy bardzo wydajnie przedstawić te fragmenty obrazu, których treść wykazuje wolnozmienny charakter. A jak już wiemy wspomniana „wolnozmienność” jest charakterystyczną cechą obrazów scen naturalnych (patrz punkt 2.3, mówiący o częstotliwościowej zawartości obrazów scen naturalnych). Do reprezentacji takich fragmentów wystarczy zaledwie kilka kosinusów o niskich częstotliwościach, o czym można się przekonać analizując zamieszczony na rysunku 2-24 przykład. W skrajnym przypadku, w którym wszystkie próbki fragmentu obrazu mają taką samą wartość (czyli fragment ten jest jednolitą teksturą), treść takiego fragmentu przekształcenie 2D-DCT „opisze” jednym niezerowym współczynnikiem $F(k_1 = 0, k_2 = 0)$, czyli kosinusem o zerowej częstotliwości (patrz zamieszczony na rysunku 2-28 przykład). Jest to możliwe, ponieważ $cos(0) = 1$. Dlatego mówi się, że **przekształcenie kosinusowe posiada zdolność skupienia dużej części informacji o sygnale w niewielkiej liczbie próbek widma**. Mówiąc inaczej, potrafi skupić energię sygnału w niewielkiej liczbie współczynników transformaty kosinusowej. Ponieważ wynikowe próbki widma (współczynniki transformaty kosinusowej) nie wykazują już tak silnego podobieństwa co wejściowe próbki obrazu (patrz rysunek 2-23), mówi się, że przekształcenie kosinusowe realizuje (przynajmniej w bardzo dużym stopniu) **dekorelację** wejściowych danych. Znaczenie owej dekorelacji danych jest wręcz ogromne z perspektywy realizowanego później na próbkach widma **entropijnego kodowania danych** (więcej na ten temat – patrz Rozdział 3).

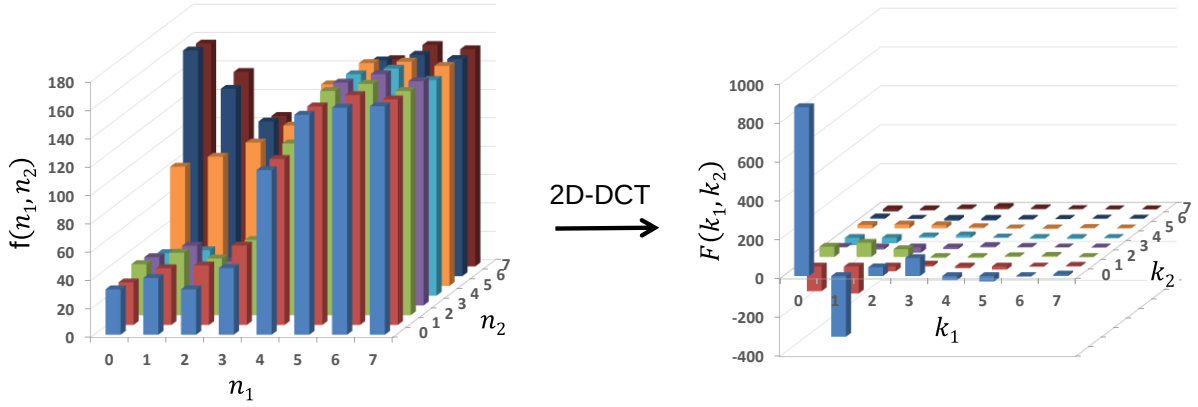
4. Funkcje bazowe przekształcenia DCT (którymi są kosinusy) są z góry znane i nie zależą od wejściowego sygnału. Nie ma zatem konieczności transmisji tych funkcji do dekodera obrazu, co jest bardzo istotne z punktu widzenia ponoszonego kosztu bitowego użycia przekształcenia.

5. Z punktu widzenia praktycznego wykorzystania przekształcenia kosinusowego nie bez znaczenie jest również istnienie szybkich algorytmów obliczania DCT. To one otworzyły drogę wdrożeniu przekształcenia do kodeków obrazu i dźwięku. Podczas gdy wyznaczenie próbek

widma wprost ze wzoru 2.20 wymagałoby aż $N_1^2 \cdot N_2^2$ operacji matematycznych[20] (czyli w przypadku obliczeń prowadzonych w blokach 8x8 daje to aż $8^2 \cdot 8^2 = 4096$ operacji w każdym bloku), to można tę wysoką złożoność przekształcenia istotnie zmniejszyć ponieważ:

a. Przekształcenie 2D-DCT jest **separowalne**. Zgodnie z tą cechą, zamiast wyznaczać w jednym kroku dwuwymiarowe przekształcenie dla bloku próbek obliczenia można rozbić na dwa etapy, w których wyznaczane są próbki przekształceń jednowymiarowych. W etapie 1 stosujemy 1D-DCT dla kolejnych wierszy bloku, a następnie (etap 2) na otrzymanym wyniku obliczamy 1D-DCT w kolejnych kolumnach (lub odwrotnie, najpierw kolumny a potem wiersze). Otrzymywany na drodze takich obliczeń wynik końcowy jest identyczny z tym uzyskiwanym dla 2D-DCT, a liczba wykonywanych operacji jest znacząco mniejsza i wynosi[21] $N_1^2 \cdot N_2 + N_2^2 \cdot N_1$ (zamiast pierwotnych $N_1^2 \cdot N_2^2$). Obliczenie próbek widma dla bloku 8x8 wiąże się zatem z kosztem $8^2 \cdot 8 + 8^2 \cdot 8 = 1024$ operacji, co stanowi 4 razy mniej obliczeń niż w przypadku bezpośredniego zastosowania wzoru na 2D-DCT.

b. Istnieją szybkie algorytmy wyznaczania współczynników przekształcenia jednowymiarowego (1D-DCT). Przykładami takich algorytmów są algorytm Chena [Chen77], algorytm Arai, Agui i Nakajimy [Arai88], czy algorytm Loefflera [Loeff89]. Z uwagi na powtarzające się wartości funkcji kosinusowych możliwe jest odpowiednie pogrupowanie wyrażeń we wzorze 2.16, co prowadzi do istotnej redukcji liczby wykonywanych operacji mnożenia i dodawania. W przypadku obliczeń prowadzonych dla 8-punktowego ($N = 8$) przekształcenia 1D-DCT z zastosowaniem algorytmu Loefflera jest to zaledwie 11 mnożeń i 29 dodawań, zamiast 64 mnożeń i 63 dodawań w przypadku zastosowania oryginalnego wzoru 2.16.



Rysunek 2-24 Przekształcenie 2D-DCT potrafi skupić znaczną część informacji o obrazie w niewielkiej liczbie współczynników $F(k_1, k_2)$. Ilustracja przytoczonej cechy przekształcenia kosinusowego.

[20] W tym przypadku (wzór na 2D-DCT) na operację matematyczną składają się: wyznaczenie wartości dwóch kosinusów, iloczyn tych wartości, następnie pomnożenie wartości próbki sygnału przez wynik iloczynu kosinusów. Dodatkowo, należy sumować wyniki cząstkowe.

[21] W przypadku realizowania obliczeń z użyciem 1D-DCT jedna operacja matematyczna to: wyznaczenie wartości kosinusa oraz iloczyn tej wartości z wartością próbki sygnału. Dodatkowo, należy sumować wyniki cząstkowe. Tak więc, jedna operacja w 1D-DCT i w 2D-DCT to nie jest dokładnie to samo (patrz poprzedni przypis).

2.8.4. Inny sposób matematycznego opisu przekształcenia DCT

W punktach 2.8.1 i 2.8.2 przekształcenie kosinusowe zostało wyrażone za pomocą tradycyjnych równań. Studiując zagadnienie DCT można się również spotkać z inną konwencją matematycznego zapisu przekształcenia, w której przekształcenie jest opisywane **równaniem macierzowym**.

Zgodnie z tym sposobem proste przekształcenie jednowymiarowe (1D-DCT) *N* wartości $f(n)$ jest zapisywane za pomocą równania:

$$\begin{bmatrix} F(0) \\ F(1) \\ \vdots \\ F(N-1) \end{bmatrix} = \begin{bmatrix} cos_{0,0} & cos_{0,1} & \cdots & cos_{0,N-1} \\ cos_{1,0} & cos_{1,1} & \cdots & cos_{1,N-1} \\ \cdots & \cdots & \cdots & \cdots \\ cos_{N-1,0} & cos_{N-1,1} & \cdots & cos_{N-1,N-1} \end{bmatrix} . \begin{bmatrix} f(0) \\ f(1) \\ \vdots \\ f(N-1) \end{bmatrix} \qquad (2.22)$$

gdzie:
$f(0), f(1), \ldots, f(N-1)$ są wartościami wejściowymi przekształcenia,
$F(0), F(1), \ldots, F(N-1)$ to wynikowe współczynniki przekształcenia kosinusowego,

natomiast macierz $\begin{bmatrix} cos_{0,0} & cos_{0,1} & \cdots & cos_{0,N-1} \\ cos_{1,0} & cos_{1,1} & \cdots & cos_{1,N-1} \\ \cdots & \cdots & \cdots & \cdots \\ cos_{N-1,0} & cos_{N-1,1} & \cdots & cos_{N-1,N-1} \end{bmatrix}$

jest **macierzą przekształcenia**[22] i określa funkcje bazowe jednowymiarowego przekształcenia kosinusowego (1D-DCT).

Mówiąc bardziej precyzyjnie, poszczególne wiersze **macierzy przekształcenia** zawierają próbki fali kosinusoidalnej o ściśle określonej częstotliwości. Zakładając przypadek 8-punktowego przekształcenia 1D-DCT (czyli $N = 8$), oraz stosując te same oznaczenia jak w punkcie 2.8.1 znaczenie poszczególnych wartości liczbowych macierzy przekształcenia jest takie jak na rysunku 2-25.

Odrobina znajomości rachunku macierzowego pozwala zauważyć, że iloczyn k-tego wiersza macierzy przekształcenia z wektorem wartości wejściowych (czyli wartości $f(n)$ dla kolejnych n) sprowadza się do przedstawionego w punkcie 2.8.1 tradycyjnego równania na $F(k)$. Widać więc, że z matematycznego punktu widzenia przedstawiony tutaj opis przekształcenia jest tożsamy z opisem zaprezentowanym w punkcie 2.8.1. Jak już wspomniano w ostatnim przypisie, w porównaniu tym pominięto wpływ występujących w równaniu 2.18 współczynników skalujących (stałych), które skalują dodatkowo próbki kosinusów. Zrobiono to po to, żeby ułatwić czytelnikowi zrozumienie prezentowanej w tym punkcie myśli. Należy jednak mocno podkreślić, że znaczenie praktyczne wspomnianego skalowania próbek kosinusów jest bardzo duże, ponieważ otrzymujemy w ten sposób **przekształcenie**, które jest **ortonormalne**, dzięki czemu:

1. Mamy możliwość oceny, jaką faktycznie część mocy wejściowego sygnału przenosi każda ze składowych harmonicznych (w tym przypadku każda z funkcji kosinusowych). Taka

[22] W praktyce elementy przedstawionej macierzy są jeszcze korygowane odpowiednimi współczynnikami skalującymi – patrz równania matematyczne przedstawione w punkcie 2.8.1. Żeby dobrze zobrazować to, czym są poszczególne wiersze tej macierzy (czyli jak je należy rozumieć) celowo zrezygnowano z przemnożenia macierzy przez wspomniane współczynniki.

wiedza jest szczególnie istotna z perspektywy stratnej kompresji obrazów, w której musimy zadecydować, z jaką dokładnością reprezentować w zakodowanym strumieniu bitowym poszczególny współczynniki przekształcenia kosinusowego (które, dla przypomnienia, odpowiadają za amplitudę i znak poszczególnych kosinusów).

2. Macierz przekształcenia odwrotnego można bardzo łatwo wyznaczyć, ponieważ jest ona transpozycją macierzy przekształcenia prostego. Dysponując więc macierzą przekształcenia po przeskalowaniu (elementy takiej macierzy oznaczymy indeksem prim: „ ' ”) przekształcenie odwrotne można zapisać jako:

$$\begin{bmatrix} f(0) \\ f(1) \\ \vdots \\ f(N-1) \end{bmatrix} = \begin{bmatrix} cos'_{0,0} & cos'_{0,1} & \dots & cos'_{0,N-1} \\ cos'_{1,0} & cos'_{1,1} & \dots & cos'_{1,N-1} \\ \dots & \dots & \dots & \dots \\ cos'_{N-1,0} & cos'_{N-1,1} & \dots & cos'_{N-1,N-1} \end{bmatrix}^T \cdot \begin{bmatrix} F(0) \\ F(1) \\ \vdots \\ F(N-1) \end{bmatrix} \tag{2.23}$$

gdzie indeks *T* oznacza transpozycję macierzy. Wzór 2.23 jest wynikiem obustronnego, lewostronnego (jest to istotne, gdyż mnożenie macierzy nie jest przemienne) przemnożenia równania przekształcenia prostego (wzór 2.22) przez macierz odwrotną do macierzy przekształcenia prostego (czyli w tym przypadku przez transponowaną macierz przekształcenia prostego).

Jak już wcześniej wspomniano, przekształcenie kosinusowe jest **separowalne**. Dwuwymiarowe przekształcenie kosinusowe (2D-DCT) otrzymamy zatem poprzez właściwe „złożenie” dwóch przekształceń jednowymiarowych, o czym była już mowa w punkcie 2.8.3.3. W zapisie macierzowym, proste przekształcenie 2D-DCT dwuwymiarowego sygnału $f(n_1, n_2)$ jest opisane równaniem:

$$\begin{aligned} \begin{bmatrix} F(0,0) & F(0,1) & \dots & F(0,N-1) \\ F(1,0) & F(1,1) & \dots & F(1,N-1) \\ \dots & \dots & \dots & \dots \\ F(N-1,0) & F(N-1,1) & \dots & F(N-1,N-1) \end{bmatrix} = \\ = \begin{bmatrix} cos'_{0,0} & cos'_{0,1} & \dots & cos'_{0,N-1} \\ cos'_{1,0} & cos'_{1,1} & \dots & cos'_{1,N-1} \\ \dots & \dots & \dots & \dots \\ cos'_{N-1,0} & cos'_{N-1,1} & \dots & cos'_{N-1,N-1} \end{bmatrix} \cdot \\ \cdot \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \dots & \dots & \dots & \dots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,N-1) \end{bmatrix} \cdot \\ \cdot \begin{bmatrix} cos'_{0,0} & cos'_{0,1} & \dots & cos'_{0,N-1} \\ cos'_{1,0} & cos'_{1,1} & \dots & cos'_{1,N-1} \\ \dots & \dots & \dots & \dots \\ cos'_{N-1,0} & cos'_{N-1,1} & \dots & cos'_{N-1,N-1} \end{bmatrix}^T \end{aligned} \tag{2.24}$$

Jako wynik otrzymujemy współczynniki $F(k_1, k_2)$ przekształcenia DCT.

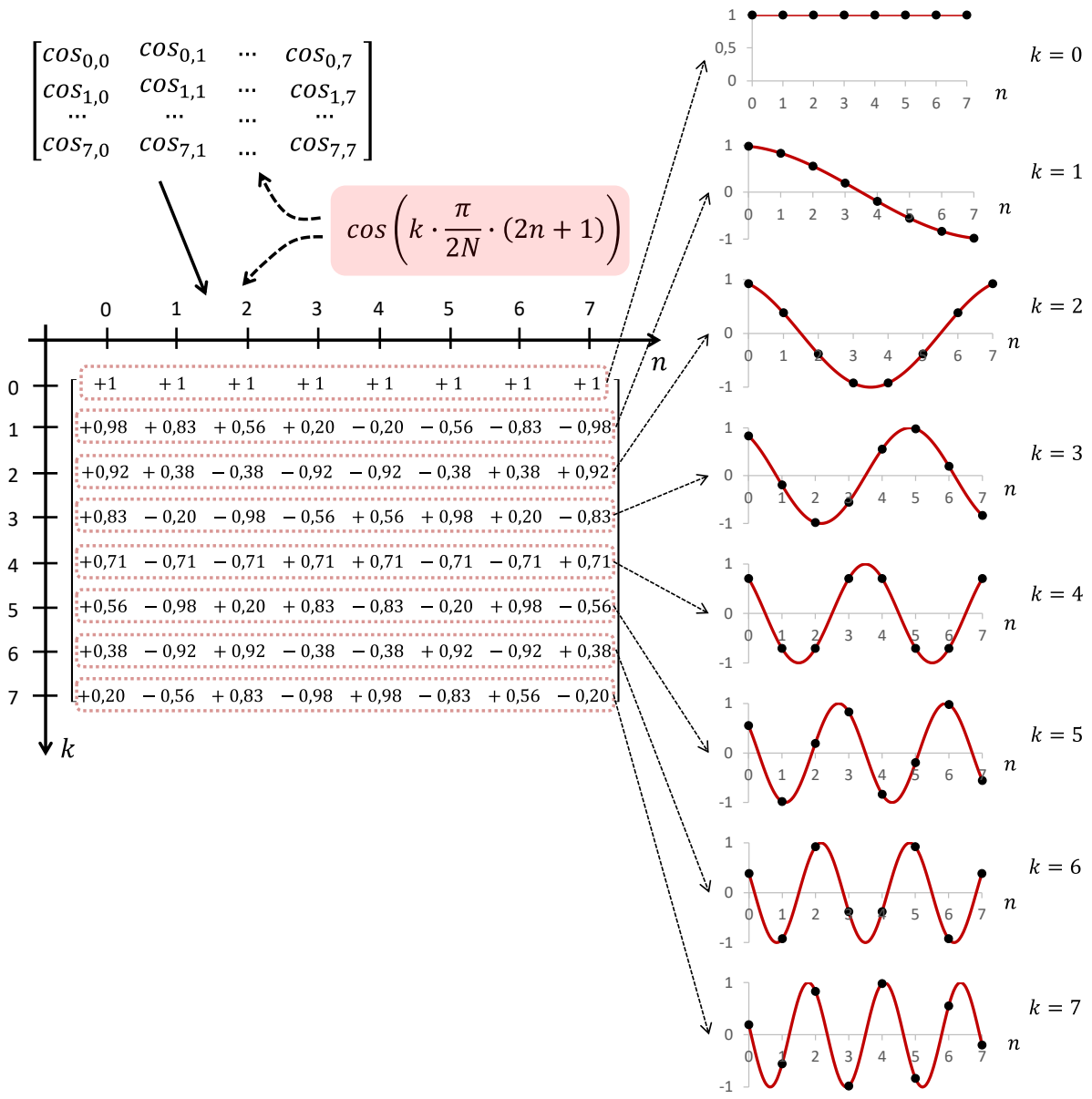
Próbki $f(n_1, n_2)$ wejściowego sygnału otrzymamy realizując przekształcenie odwrotne, zgodnie z poniższym równaniem:

$$\begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \dots & \dots & \dots & \dots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,N-1) \end{bmatrix} =$$

$$= \begin{bmatrix} cos'_{0,0} & cos'_{0,1} & \dots & cos'_{0,N-1} \\ cos'_{1,0} & cos'_{1,1} & \dots & cos'_{1,N-1} \\ \dots & \dots & \dots & \dots \\ cos'_{N-1,0} & cos'_{N-1,1} & \dots & cos'_{N-1,N-1} \end{bmatrix}^{T} \cdot$$

$$\cdot \begin{bmatrix} F(0,0) & F(0,1) & \dots & F(0,N-1) \\ F(1,0) & F(1,1) & \dots & F(1,N-1) \\ \dots & \dots & \dots & \dots \\ F(N-1,0) & F(N-1,1) & \dots & F(N-1,N-1) \end{bmatrix} \cdot \tag{2.25}$$

$$\cdot \begin{bmatrix} cos'_{0,0} & cos'_{0,1} & \dots & cos'_{0,N-1} \\ cos'_{1,0} & cos'_{1,1} & \dots & cos'_{1,N-1} \\ \dots & \dots & \dots & \dots \\ cos'_{N-1,0} & cos'_{N-1,1} & \dots & cos'_{N-1,N-1} \end{bmatrix}$$



Rysunek 2-25 Znaczenie wartości elementów **macierzy przekształcenia kosinusowego** (1D-DCT). Na rysunku założono przypadek przekształcenia 8-punktowego ($N = 8$).

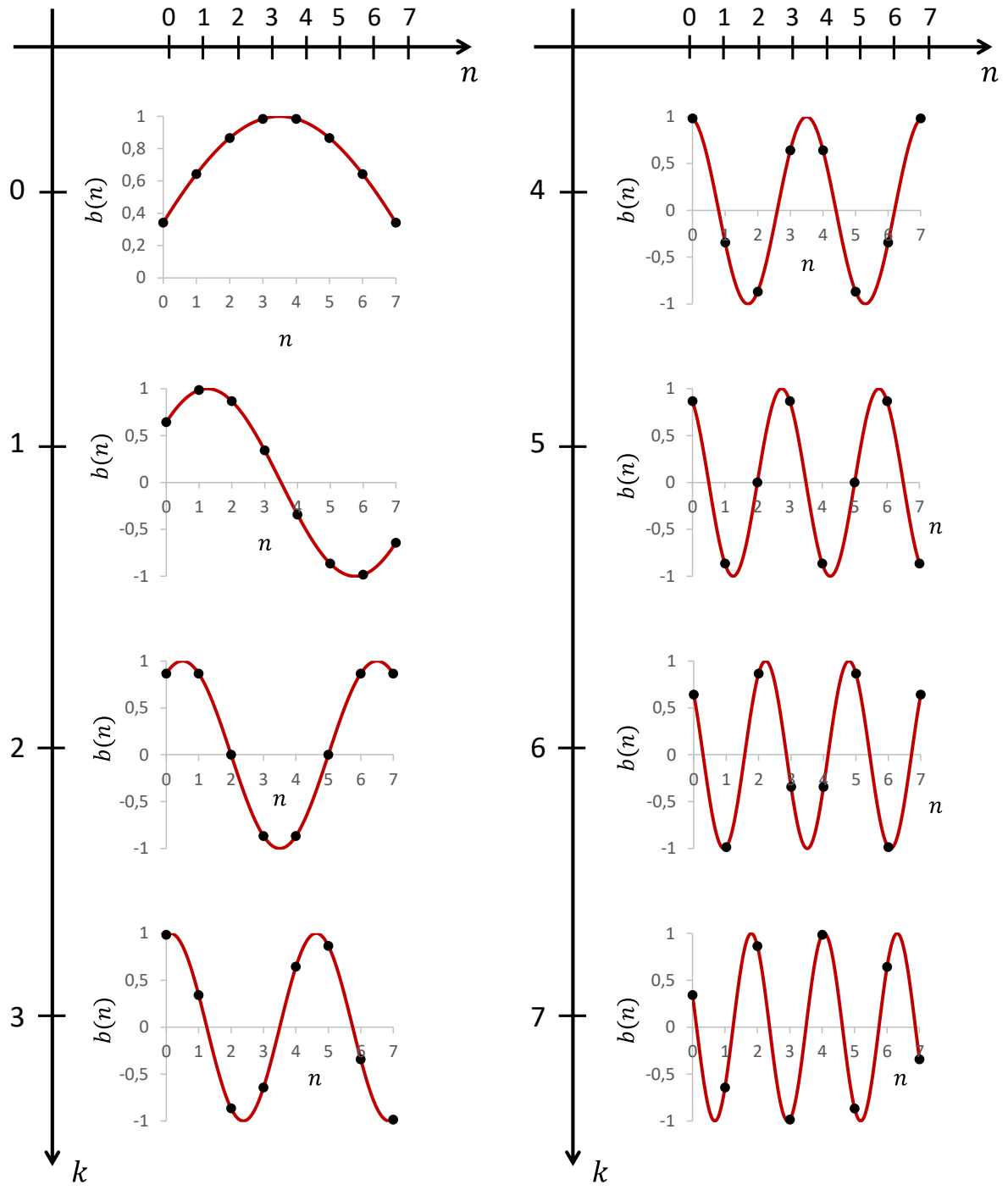
2.8.5. Dlaczego nie przekształcenie sinusowe zamiast kosinusowego?

Wiemy już, że opis obrazów za pomocą kosinusów prowadzi do bardzo wydajnej jego reprezentacji. Jednak co by było, gdybyśmy zamiast kosinusów zastosowali **funkcje sinusoidalne**? Czy w takim przypadku reprezentacja obrazów byłaby równie wydajna? Żeby odpowiedzieć na tak postawione pytania należy przeanalizować „wygląd” sinusów, które stanowią podstawę przekształcenia znanego w literaturze jako **dyskretne przekształcenie sinusowe** (ang. Discrete Sine Transformation – DST) [Salom10].

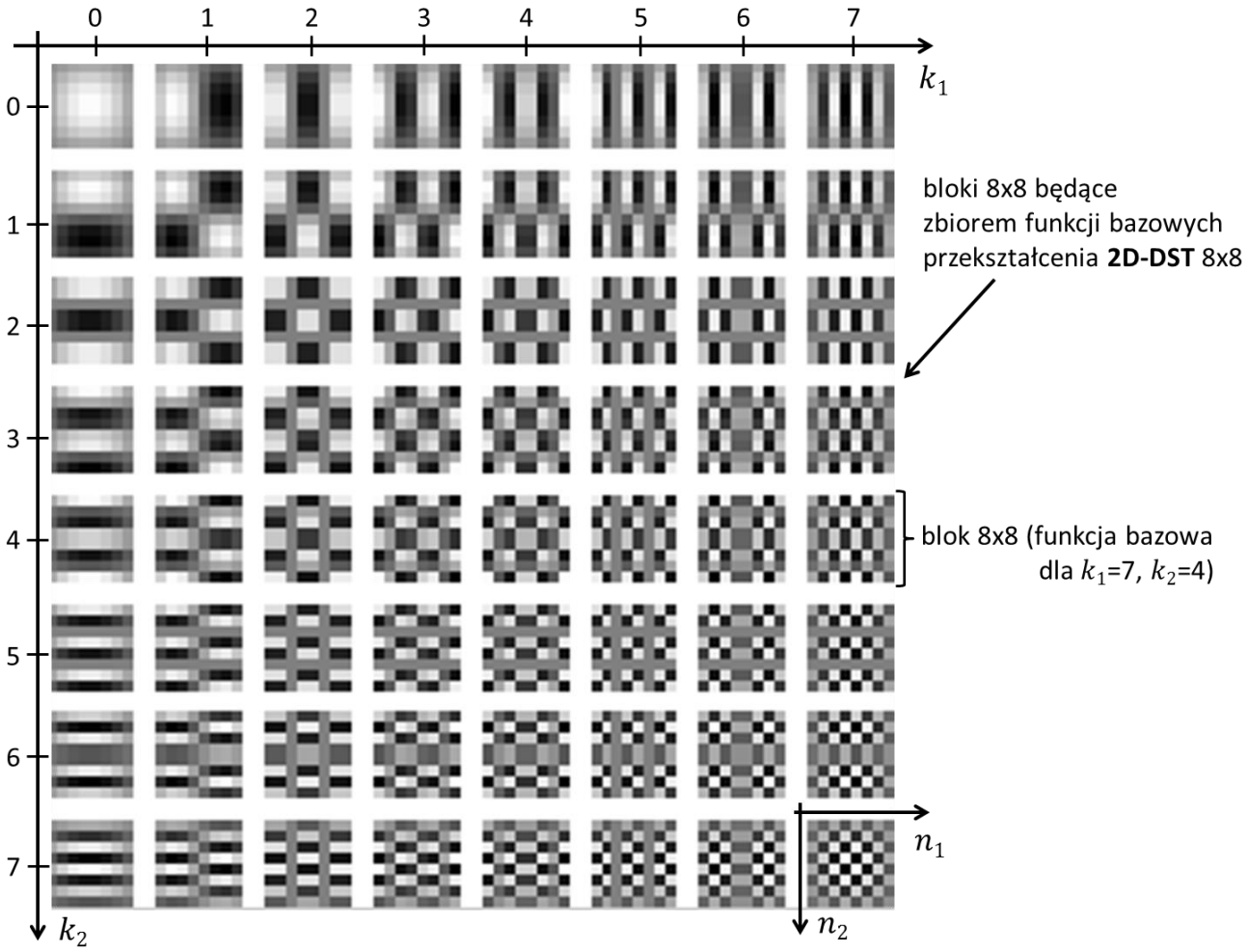
Kolejne funkcje jednowymiarowego przekształcenia DST (czyli 1D-DST) zostały przedstawione na rysunku 2-26. Rysunek 2-27 prezentuje z kolei funkcje przekształcenia dwuwymiarowego (czyli 2D-DST), z których każda jest wynikiem iloczynu dwóch właściwych

funkcji sinusoidalnych jednowymiarowych. Analizując „wygląd” tych funkcji należy zauważyć, że w ich zbiorze nie ma funkcji, której wszystkie próbki przyjmują jednakową wartość, czyli funkcji, której częstotliwości przestrzenne pozioma oraz pionowa są zerowe. Przypomnijmy, że obecność takiej funkcji w przekształceniu kosinusowym daje możliwość opisywania nieskomplikowanych fragmentów obrazu (których jest w obrazie naturalnym najwięcej) przy pomocy małej liczby niezerowych próbek widma. W przypadku zupełnie gładkiego obszaru obrazu jest to zaledwie jedna próbka widma (patrz rysunek 2-28). Z uwagi na brak tej składowej w zbiorze funkcji przekształcenia DST nie wykazuje ono, w porównaniu z DCT, aż tak dużej zdolności skupiania informacji o wolnozmiennym sygnale w małej liczbie próbek widma. W scenariuszu, który jest najbardziej pomyślny z perspektywy przekształcenia 2D-DCT (reprezentacja gładkiego fragmentu obrazu jedną próbką widma), przekształcenie sinusowe będzie potrzebować tych próbek widma więcej (patrz zamieszona na rysunku 2-28 ilustracja). Stąd w ogólności niższa niż w przypadku przekształcenia 2D-DCT, wydajność transformacji 2D-DST w zadaniach kompresji obrazów.

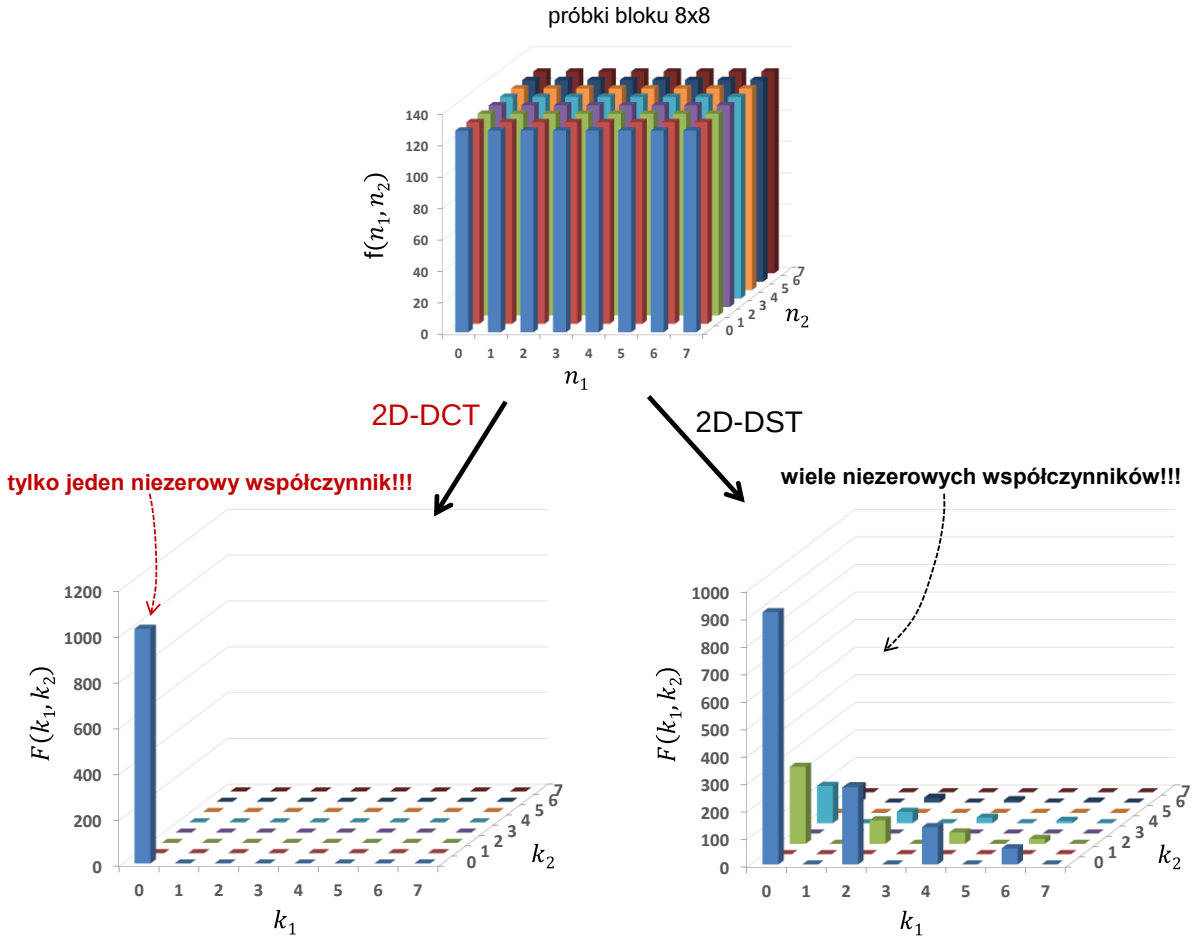
Badania pokazują jednak, że postać funkcji przekształcenia sinusowego predysponuje je do wydajniejszej niż w przypadku funkcji kosinusowych, reprezentacji tych fragmentów obrazu, których wartości próbek nie wykazują aż tak silnego podobieństwa, czyli są słabiej ze sobą skorelowane. W przypadku zatem bardziej złożonych fragmentów obrazu (czyli z większym zróżnicowaniem wartości próbek) uzasadniona może być ich reprezentacja w dziedzinie współczynników przekształcenia 2D-DST. Z tej perspektywy mówi się, że przekształcenie sinusowe jest komplementarne do przekształcenia kosinusowego – tam gdzie lepiej radzi sobie jedno przekształcenie gorzej wypada to drugie i odwrotnie. Z tego powodu w najnowszych koderach obrazu (np. koder MPEG-H HEVC/H.265), obok stosowania przekształcenia 2D-DCT stosuje się również transformację sinusową w określonych trybach kodowania fragmentów obrazu.



Rysunek 2-26 Zestaw **jednowymiarowych sinusów** będących zbiorem funkcji bazowych przekształcenia 1D-DST dla $N = 8$. W tym przypadku zestaw składa się z 8 jednowymiarowych sinusów (o różnych częstotliwościach), przy czym każdy z sinusów składa się z 8 próbek. W drodze ważonej sumy powyższych sinusów reprezentowany jest wejściowy sygnał. Obrazy sinusów przygotowano z użyciem własnego oprogramowania. Sinusy wyznaczono ze wzoru: $b(n) = sin\big(\pi \cdot (n+1) \cdot (k+1)/(N+1)\big)$.



Rysunek 2-27 Zestaw **dwuwymiarowych sinusoid** będących zbiorem funkcji bazowych przekształcenia 2D-DST dla $N_1 = 8$, $N_2 = 8$. W tym przypadku zestaw składa się z 64 dwuwymiarowych sinusoid (o różnych częstotliwościach), przy czym każda sinusoida jest obrazkiem o wielkości 8x8 próbek. W drodze ważonej sumy powyższych sinusów możliwe jest odwzorowanie treści wejściowego obrazu. Obrazy przygotowano z użyciem własnego oprogramowania.



Rysunek 2-28 W porównaniu z przekształceniem sinusowym **przekształcenie kosinusowe** posiada większą zdolność skupiania informacji o obrazie w małej liczbie próbek widma. Tak jest w przypadku reprezentacji obrazów, w których wartości sąsiednich próbek wykazują silne podobieństwo (czyli wysoka korelacja wartości próbek). W tym przykładzie wartości wszystkich próbek f są takie same.

2.8.6. Sposób użycia 2D-DCT w koderach obrazu

Przekształcenie 2D-DCT powinno być w koderze obrazu stosowane w sposób, który będzie uwzględniał dwie następujące kwestie:

1. Wymagane do realizacji przekształcenia nakłady obliczeniowe procesora (lub innego układu liczącego), jak również wymagania algorytmu na pamięć.

2. Zdolność przekształcenia do reprezentacji wejściowych wartości przy pomocy możliwie małej liczby niezerowych próbek widma.

Właściwe uwzględnienie przytoczonych powyżej założeń umożliwi optymalny, czyli najlepszy możliwy, sposób praktycznej aplikacji 2D-DCT w koderach obrazów.

Prawdopodobnie najprostszym koncepcyjnie podejściem byłoby stosowanie przekształcenia kosinusowego bezpośrednio na treści całego obrazu. Jednak taki sposób użycia przekształcenia wiązałby się z bardzo dużą złożonością algorytmu. W przypadku obrazu

składającego się z $N_1 x\, N_2$ próbek realizacja 2D-DCT wprost ze wzoru 2.18, z uwzględnieniem separowalności transformacji, wymaga $N_1^2 \cdot N_2 + N_2^2 \cdot N_1$ operacji matematycznych[23]. Nie trudno jest zatem policzyć, że gdybyśmy operowali na obrazie o wysokiej rozdzielczości, np. 2048 x 2048 próbek to dałoby to ogromną liczbę 17,18 miliarda wykonywanych operacji! W tym konkretnym przykładzie konieczne byłoby zapamiętanie niemałej liczby wartości definiujących próbki kosinusów jednowymiarowego przekształcenie DCT (w sumie 2048 x 2048 liczb, bo 2048 kosinusów, gdzie każdy z nich jest opisany przy pomocy 2048 próbek), co oczywiście przekłada się na określone wymagania pamięciowe tego algorytmu.

Widać więc, że utrzymanie złożoności obliczeniowej i pamięciowej algorytmu w rozsądnych granicach wymaga jego stosowania na mniejszych zbiorach wartości. Z tej perspektywy, zamiast w jednym kroku liczyć 2D-DCT na całym obrazie lepiej jest podzielić obraz na niezależne, nieduże fragmenty i prowadzić obliczenia w tych fragmentach. Dla przykładu, gdyby jednym takim fragmentem był mały blok obrazu o wielkości 8x8 próbek, to wykonanie obliczeń w takim bloku sprowadzałoby się do wykonania 1024 operacji, co biorąc pod uwagę liczbę 65 536 bloków 8x8 w całym obrazie o wielkości 2048 x 2048 dałoby w sumie 67,1 miliona operacji. Czyli tych operacji byłoby 256 razy mniej niż w przypadku operowania na całym obrazie. Nieporównywalnie mniejszy byłby również pamięciowy koszt zapamiętania próbek kosinusów przekształcenia 1D-DCT, co łatwo można policzyć.

W tym miejscu kluczowe jest jednak następujące pytanie, czy operowanie na niedużych blokach obrazu będzie również korzystne z punktu widzenia efektywności reprezentacji danych przez 2D-DCT? Odpowiedź na to pytanie można wyczytać z zaprezentowanej w punktach 2.8.3.3 oraz 2.8.5 analizy możliwości przekształcenia 2D-DCT w zadaniu reprezentacji danych.

Z analizy tej wprost widać, że im większe podobieństwo (większa korelacja) kolejnych wartości próbek, które mają być przedmiotem przekształcenia kosinusowego, tym większa zdolność tego przekształcenia do reprezentacji tych wartości (próbek) przy pomocy małej liczby niezerowych próbek widma. Oczywiście, w przypadku niewielkiej liczby kolejnych próbek obrazu zachodzi większa szansa na silne podobieństwo ich wartości niż w sytuacji bardzo licznego zbioru próbek, który opisuje np. cały wiersz próbek w obrazie. Przekształcenie 2D-DCT powinno się zatem stosować dla bloków o takim, zwykle niewielkim rozmiarze, który ciągle gwarantuje duże podobieństwo wartości znajdujących się w nim próbek. W ten sposób zapewnimy przekształceniu warunki pracy, w których wykazuje ono najwyższą efektywność reprezentacji danych. Proszę zauważyć, że najlepszy dla osiągnięcia tego celu rozmiar bloku obrazu będzie silnie zależał od treści obrazu, ale również od jego przestrzennej rozdzielczości.

Należy tutaj zauważyć, że wskazana w zdaniu poprzednim celowość stosowania 2D-DCT na blokach próbek o niewielkim rozmiarze pozostaje w pełnej zgodzie z wytycznymi dotyczącymi złożoności algorytmu (mała złożoność w przypadku operowania na małych blokach próbek).

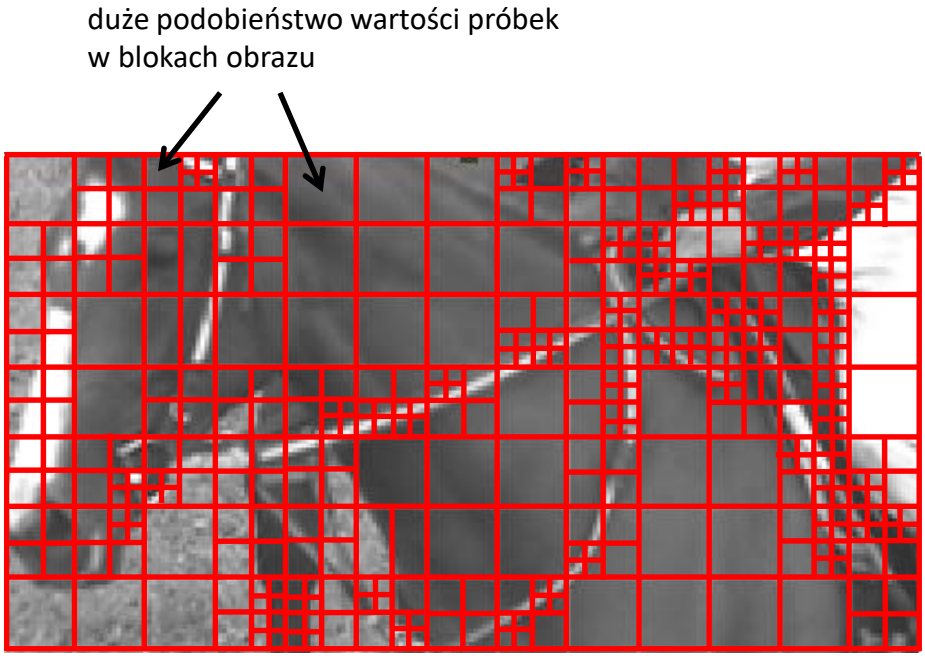
Uwzględniając powyższe wnioski starsze techniki kompresji obrazów (np. JPEG, MPEG-2) realizowały przekształcenie kosinusowe (dwuwymiarowe) w blokach obrazu o stałej wielkości 8x8 próbek. Opracowanie wymienionych technik przypadło na lata (pierwsza połowa lat 90-tych), w których na skalę masową operowano na takich obrazach, których przestrzenna rozdzielczość nie była większa niż standardowa (czyli, np. 704x576 próbek obrazu). Z perspektywy obrazów o takiej niewielkiej rozdzielczości oraz uwzględniając dodatkowo aspekt obliczeniowej złożoności przekształcenia i dostępnej wówczas mocy obliczeniowej układów liczących wypracowanym

[23] W przypadku realizowania obliczeń z użyciem 1D-DCT jedna operacja matematyczna to: wyznaczenie wartości kosinusa oraz iloczyn tej wartości z wartością próbki sygnału. Dodatkowo, należy sumować wyniki cząstkowe.

kompromisem był właśnie rozmiar 8x8 dla bloku obrazu. Oczywistym uproszczeniem pozostawał jednak stały rozmiar stosowanych bloków.

Z biegiem lat, wraz z rozwojem technologii rejestracji i kompresji obrazów, przestrzenna rozdzielczość stosowanych obrazów istotnie się zwiększyła. Z punktu widzenia oczekiwań zwykłego odbiorcy standardem jest dzisiaj (rok 2018) obraz w tzw. rozdzielczości Full HD, czyli obraz składający się z 1920x1080 próbek obrazu. Ponieważ ta wyższa rozdzielczość obrazu jest wynikiem gęstszego próbkowania przez kamerę widoku rejestrowanej sceny, to w takim obrazie większa niż w przypadku obrazu o niskiej rozdzielczości, liczba kolejnych próbek może wykazywać wysokie podobieństwo. Wyższa rozdzielczość obrazów uzasadnia więc realizację 2D-DCT na blokach większych niż pierwotnie stosowany rozmiar 8x8.

Powyższe obserwacje doczekały się już praktycznej realizacji w koderach obrazu. Najnowsze techniki kompresji (np. MPEG-H HEVC/H.265) realizują przekształcenie kosinusowe w blokach o rozmiarach od 4x4 do 32x32 próbki obrazu. W tych technikach, wybór optymalnego rozmiaru bloku jest adaptacyjny i zależy od lokalnej treści obrazu, która jest zawarta w tym bloku. Przypomnijmy raz jeszcze – dążymy do realizacji 2D-DCT w blokach o takim rozmiarze, w których wartości kolejnych próbek ciągle wykazują silne podobieństwo. Przeprowadzone na sekwencjach o rozdzielczości Full HD badania pokazały, że dla przekształcenia 2D-DCT wspomniany zakres rozmiaru bloku (od 4x4 do 32x32) jest dobrym kompromisem pomiędzy efektywnością reprezentacji obrazów i złożonością obliczeniową przekształcenia. Jednak wraz z kolejnym wzrostem rozdzielczości kodowanych obrazów w niedalekiej już przyszłości należy się spodziewać dalszego wzrostu rozmiaru bloków, dla których zasadne okaże się stosowanie przekształcenia 2D-DCT.



Rysunek 2-29 Ilustracja przesłanki dla doboru rozmiaru bloku obrazu, w którym warto wyznaczać próbki przekształcenia 2D-DCT. Powinno się to robić w blokach o takim rozmiarze, w których wartości kolejnych próbek ciągle wykazują silną korelację (silne podobieństwo wartości).

2.9. Inne stosowane przekształcenie – transformacja Walsha-Hadamarda

Wyliczenie próbek widma przekształcenia kosinusowego (sinusowego również) wymaga wykonania określonej liczby operacji pomnożenia wartości próbek sygnału (obrazu) przez wartości odpowiednich funkcji trygonometrycznych oraz dodania do siebie cząstkowych rezultatów obliczeń. W literaturze dobrze znane jest inne przekształcenie, nazywane **transformacją Walsha-Hadamarda** (ang. Walsh-Hadamard Transformation – WHT) [Salom10], które umożliwia prostszą obliczeniowo (w porównaniu z przekształceniami kosinusowym i sinusowym) analizę częstotliwościową wejściowego sygnału (obrazu).

Podstawą tego przekształcenia są przebiegi prostokątne o różnych częstotliwościach zmian wartości próbek. W wersji jednowymiarowej (przekształcenie 1D-WHT) oraz dwuwymiarowej (przekształcenie 2D-WHT) przebiegi te zostały zobrazowane na rysunkach 2-30 i 2-31. Ponieważ dla danego argumentu funkcja prostokątna przyjmuje tylko jedną z dwóch wartości: +1 lub -1, to czynność wyznaczenia próbek widma z użyciem WHT jest znacznie prostsza. Sprowadza się ona do wykonania samych operacji dodawania i odejmowania wartości, już bez operacji mnożenia liczb. Niższa złożoność WHT uzasadniałaby szerokie praktyczne stosowanie tego przekształcenia w samej kompresji obrazów, jednak eksperymentalnie wykazano, że zdolności WHT do wydajnej reprezentacji treści obrazów są w ogólności niższe niż te dla DCT. Wyjątkiem są tutaj dane wytwarzane przez nieliczne tryby kompresji, dla których w najnowszych koderach (np. MPEG-4 AVC/H.264) faktycznie stosuje się WHT zamiast DCT.

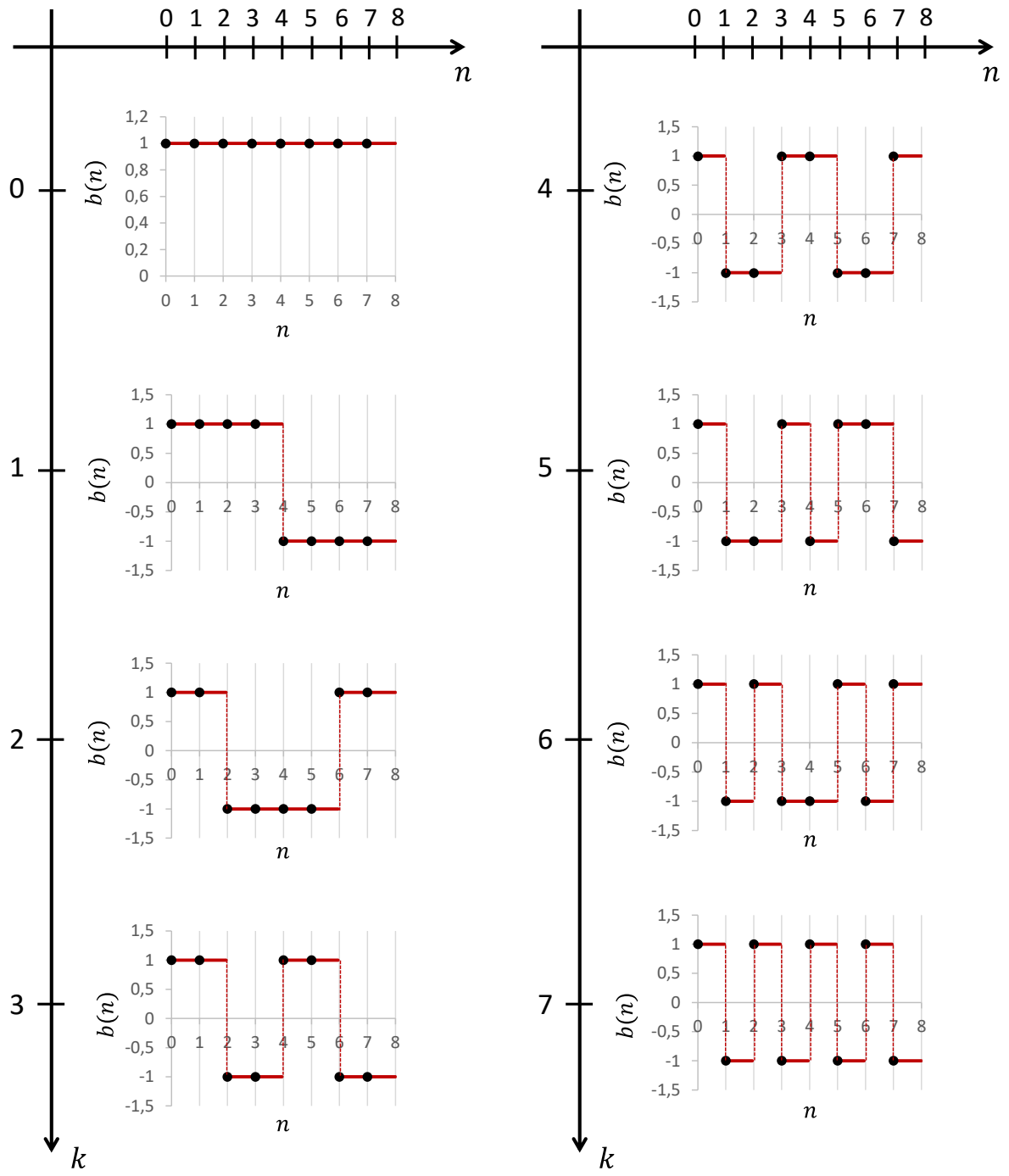
Nie oznacza to jednak, że zastosowanie w koderach obrazu przekształcenia WHT jest zupełnie marginalne. W praktyce kompresji zakodowanie fragmentu obrazu jest poprzedzone wyborem najlepszego trybu kompresji (czyli trybu, który przy możliwie małej liczbie bitów pozwala uzyskać określoną jakość zakodowanego fragmentu obrazu). Żeby ten najlepszy tryb wybrać, konieczne jest wykonanie wielokrotnej kompresji fragmentu, za każdym razem z użyciem innego dostępnego trybu i porównanie ze sobą otrzymanych wyników kodowania. Ponieważ w najnowszych koderach pula dostępnych trybów jest bardzo duża, to testowanie danego trybu na drodze realizacji pełnego kodowania z jego użyciem wiązałoby się z ogromną wręcz złożonością obliczeniową (czyli bardzo długim czasem kodowania obrazów).

Dlatego, rynkowa praktyka stosowania koderów wymaga wprowadzenia istotnych uproszczeń do mechanizmu wyboru trybów kompresji. W ramach tychże uproszczeń złożoną procedurę pełnego testowania kolejnych trybów można zastąpić następującymi prostszymi rozwiązaniami:

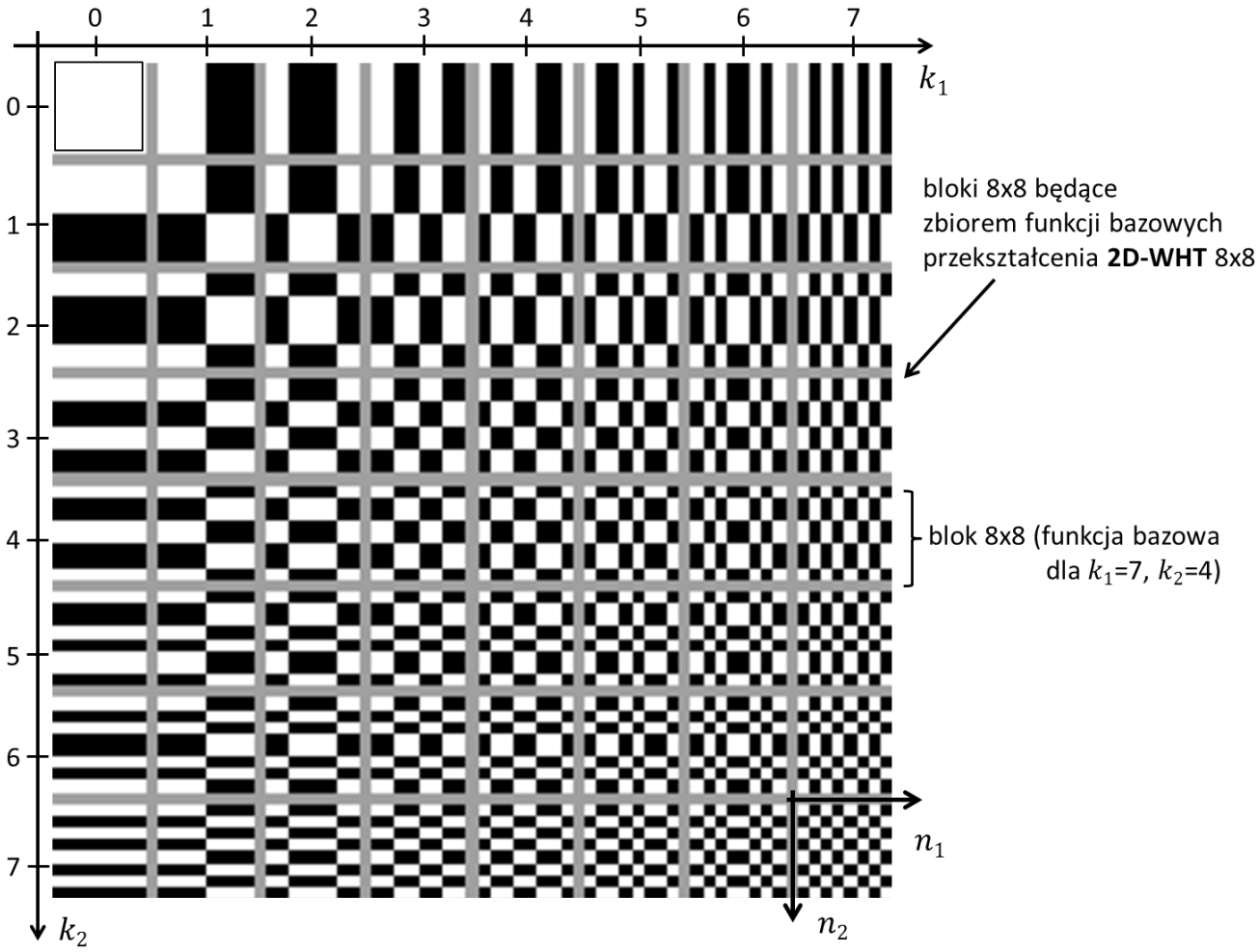
1. Na etapie wyboru trybów DCT można zastąpić prostszym przekształceniem WHT. Takie rozwiązanie stosuje się bardzo często.

2. Testowanie trybów można poprzedzić uproszczoną analizą częstotliwościową kodowanej treści, realizowaną właśnie w oparciu o przekształcenie WHT. Wyniki tej analizy częstotliwościowej mogą być później podstawą do wykluczenia z procedury testowania wybranych trybów kompresji. Jest to również bardzo często stosowane rozwiązanie.

Współczesne kodery obrazu wykorzystują zatem różne rodzaje transformacji danych wejściowego obrazu i czynią to w sposób uwzględniający dobrze już poznane zalety tych przekształceń. W omawianym przypadku, DCT wydajnie redukuje statystyczną nadmiarowość wejściowych danych, więc jest stosowane podczas finalnej kompresji fragmentu obrazu, realizowanej z użyciem wybranego trybu kodowania. WHT cechuje natomiast krótki czas obliczeń, co uzasadnia jego zastosowanie w procedurze wyboru trybów kompresji.



Rysunek 2-30 Zestaw **jednowymiarowych przebiegów prostokątnych** będących zbiorem funkcji bazowych przekształcenia 1D-WHT dla $N = 8$. W tym przypadku zestaw składa się z 8 jednowymiarowych funkcji prostokątnych (o różnych częstotliwościach zmian wartości), przy czym każda z funkcji jest reprezentowana przy pomocy 8 próbek. Wejściowy sygnał jednowymiarowy można reprezentować jako sumę ważoną powyższych funkcji.



Rysunek 2-31 Zestaw **dwuwymiarowych funkcji prostokątnych** będących zbiorem funkcji bazowych przekształcenia 2D-WHT dla $N_1 = 8, N_2 = 8$. W tym przypadku zestaw składa się z 64 przebiegów (o różnych częstotliwościach), przy czym każda z funkcji jest obrazkiem o wielkości 8x8 próbek. W drodze ważonej sumy powyższych funkcji możliwe jest odwzorowanie treści wejściowego obrazu. Obrazy przygotowano z użyciem własnego oprogramowania.

2.10. Przekształcenie „najlepsze z najlepszych”, czyli transformacja Karhunena-Loévego

Przekształcenie DCT realizuje bardzo wydajną reprezentację częstotliwościową fragmentów obrazu naturalnego, pomimo że korzysta ono z uniwersalnego, tzn. niezależnego od postaci wejściowego sygnału, zbioru funkcji bazowych (tymi funkcjami są oczywiście kosinusy). Istnieje jednak możliwość dalszego (chociaż już relatywnie niewielkiego) zwiększenia wspomnianej wydajności, jeśli w miejsce tych funkcji uniwersalnych zastosuje się nowe funkcje, które są dedykowane do reprezentowania takiego typu sygnału jak ten dla którego wyznacza się współczynniki przekształcenia. Te nowe funkcje prowadzą do nowego przekształcenia, które jest w literaturze znane jako **transformacja Karhunena-Loévego** (ang. Karhunen-Loéve Tranformation – KLT) lub **transformacja Hotellinga** [Salom10].

W przypadku tej nowej transformacji postać funkcji bazowych, czyli funkcji, przy pomocy których będziemy reprezentować fragment naszego obrazu, zależy bezpośrednio od postaci próbek

kodowanego obrazu (lub jakiejś części tych próbek) [24]. Jest to więc transformacja zależna od sygnału (czego nie można powiedzieć o wcześniej omówionych przekształceniach, w tym o transformacji DCT). Z matematycznego punktu widzenia kolejne funkcje bazowe KLT to **wektory własne macierzy kowariancji** wejściowego zbioru próbek. W związku z tym faktem, właściwa transformacja sygnału musi być poprzedzona etapem wyznaczenia właściwych funkcji bazowych przekształcenia, co ma niekorzystny wpływ na złożoność algorytmu. W przypadku operowania na sygnale, którego charakter nie jest stały, ale się zmienia (a tak jest w przypadku obrazu – poszczególne fragmenty obrazu mogą się różnić, ponieważ mogą reprezentować treść o innym charakterze) funkcje bazowe przekształcenia KLT należałoby na nowo wyliczać. Jednak nagrodą za te dodatkowe obliczenia są nieskorelowane (lub bardzo, bardzo słabo skorelowane) współczynniki transformacji, czyli próbki przekształcenia, których wartości nie wykazują już niemal żadnego podobieństwa. Przekształcenie KLT dokonuje więc najlepszej dekorelacji wejściowych wartości, przez co w najlepszy możliwy sposób skupia ono znaczną część informacji o wejściowym sygnale w małej liczbie współczynników przekształcenia. Śmiało można więc powiedzieć, że zastosowanie KLT prowadzi do najlepszych rezultatów późniejszej reprezentacji wynikowych danych przekształcenia, jeśli wynik ten porówna się z tym, co oferują inne przekształcenia (w tym DCT).

Jednak wspomniana powyżej najlepsza wydajność przekształcenia KLT jest z perspektywy wielu praktycznych zastosowań czysto teoretyczna. Wyznaczony dla aktualnie przetwarzanego fragmentu obrazu dedykowany zbiór funkcji bazowych, który będzie opisywał poszczególne bloki tego fragmentu, należy wysłać do dekodera, wraz z rezultatem przekształcenia otrzymanego w blokach. Ponieważ w obrazie naturalnym treść kolejnych jego fragmentów może się od siebie różnić (choćby nieznacznie), to wspomnianą transmisję do dekodera nowo wyliczonych funkcji bazowych należałoby wykonywać relatywnie często. Praktyka pokazuje, że związany z tą transmisją koszt bitowy jest na tyle wysoki, że niweluje osiągniętą wcześniej wysoką wydajność reprezentacji danych wejściowych, a nawet wprowadza pewną stratę w stosunku do np. wyników z DCT. Dlatego KLT nie znajduje szerokiego zastosowania praktycznego w koderach obrazu. Rzadkie przypadki aplikacji KLT dotyczą pojedynczych trybów kompresji w najnowszych koderach, bądź użycia KLT do reprezentacji danych, których statystyka zmienia się bardzo, bardzo powoli. Z uwagi natomiast na swoje cechy KLT stanowi znakomity punkt odniesienia w badaniach wydajności innych metod kodowania transformatowego danych.

Jednak nawet w przypadkach uzasadnionego zastosowania KLT wciąż doskwierał nam będzie problem niemałych nakładów obliczeniowych, które przyjdzie nam ponosić w związku z częstym wyznaczaniem nowych funkcji bazowych przekształcenia.

2.11. Reprezentacja obrazu w dziedzinie częstotliwości – dlaczego warto?

Przedstawiona w tym rozdziale idea częstotliwościowej reprezentacji obrazu ma kluczowe znaczenie dla wydajnej kompresji obrazu, co było już mocno akcentowane w wielu miejscach poprzednich punktów. Celem pewnego usystematyzowania oraz podsumowania przedstawionych już wcześniej przemyśleń, zostaną one ponownie przywołane również w tym punkcie.

Z uwagi na bardzo duże podobieństwo wartości sąsiednich próbek w obrazie naturalnym, ich niezależne kodowanie wiązałoby się z istotną nadmiarowością bitową reprezentacji danych. Bardzo podobne do siebie wartości kodowalibyśmy niejako wielokrotnie. Zamiast więc takiego

[24] W praktyce potrzebujemy pewnego zbioru próbek (wartości), żeby na ich podstawie wyznaczyć funkcje bazowe przekształcenia. Ten zbiór próbek „uczących” jest większy od liczby próbek w pojedynczym bloku obrazu, dla którego będzie później liczone przekształcenie.

sposobu kodowania informacji o obrazie, znacznie korzystniej jest najpierw sprawdzić, jakie składowe harmoniczne zawarte są w treści obrazu, żeby następnie wysłać informację o tych składowych do dekodera obrazu. Praktyka pokazuje, że wspomnianą analizę zawartości w obrazie składowych harmonicznych warto jest przeprowadzić z użyciem przekształcenia 2D-DCT, co zostało już wcześniej poparte względami merytorycznymi. W tym przypadku obraz jest reprezentowany przy pomocy **dwuwymiarowych funkcji kosinusoidalnych** o z góry znanej wartości fazy początkowej. Ponieważ ogólna postać tych funkcji jest dobrze znana (również po stronie dekodera), to wystarczy do dekodera wysłać zaledwie informację o amplitudzie poszczególnych kosinusów, oraz poinformować o znaku funkcji (+ lub –). W drodze ważonej sumy „przesłanych” w ten sposób kosinusów dekoder odtworzy treść zakodowanego wcześniej obrazu. W finalnym rozrachunku koszt bitowy transmisji wspomnianych danych jest dużo mniejszy niż ten wynikający z bezpośredniego kodowania próbek obrazu.

Jednak to nie są jeszcze wszystkie korzyści płynące z użycia reprezentacji częstotliwościowej obrazu dla celów jego kompresji. Przedstawiając nasz obraz w domenie częstotliwości zyskujemy dodatkowo możliwość uwzględnienia na etapie kompresji obrazu dobrze znanych zdolności i ograniczeń aparatu widzenia człowieka. Zgodnie z nimi, zdolności percepcji przez człowieka niskoczęstotliwościowych składowych obrazu (którym odpowiadają treści obrazu pozbawione dużej ilości szczegółów) są większe niż składowych wysokoczęstotliwościowych (którym odpowiada skomplikowana treść obrazu). Mówiąc inaczej, możliwości odbioru przez człowieka złożonych fragmentów obrazu (detali) są znacznie ograniczone. Powyższe obserwacje uzasadniają opisywanie składowych harmonicznych nisko- i wysokoczęstotliwościowych z różną, a nie jednakową dokładnością. Składowe harmoniczne o częstotliwościach niskich należy kodować z większą dokładnością, podczas gdy składowe o częstotliwościach wysokich można reprezentować bardziej zgrubnie. W praktyce, wspomnianą niejednorodność dokładności reprezentacji poszczególnych składowych uzyskuje się w koderze poprzez zastosowanie właściwie określonej operacji **kwantowania** próbek widma obrazu (czyli kodowanie stratne próbek widma). Namiastkę efektów takiego podejścia można zobaczyć porównując zamieszczone na rysunku 2-32 obrazy, gdzie zastosowano koncepcyjnie bardzo proste rozwiązanie, polegające na usunięciu z bloków obrazu wybranych próbek przekształcenia 2D-DCT. Jednak jeszcze lepsze rezultaty można uzyskać wprowadzając bardziej zaawansowany (niż przyjęty na poniższym rysunku) mechanizm kwantowania próbek widma. Nie ulega więc wątpliwości, iż częstotliwościowa reprezentacja obrazu, w połączeniu z ideą stratnego kodowania danych, pozwala w przypadku obrazów scen naturalnych uzyskać bardzo dobry kompromis „koszt bitowy zakodowanych danych obrazu – jakość dekodowanego obrazu”.

Proszę zauważyć, że takie zróżnicowanie dokładności reprezentacji poszczególnych danych obrazu nie jest możliwe (przynajmniej w tak dużym zakresie) w przypadku kompresji przeprowadzanej w dziedzinie próbek obrazu. Z wymienionych powodów realizowana w dziedzinie częstotliwości obrazu kompresja danych stanowi podstawę wielu współczesnych sposobów reprezentacji danych obrazowych.

obraz oryginalny

obraz odtworzony

(na podstawie wybranych niskoczęstotliwościowych współczynników 2D-DCT, stanowiących 23% wszystkich współczynników)

Rysunek 2-32 Po lewej – obraz oryginalny, po prawej – obraz powstały przez proste usunięcie z częstotliwościowej reprezentacji kolejnych bloków 8x8 obrazu oryginalnego 77% współczynników przekształcenia 2D-DCT. Usuwanymi współczynnikami były te związane ze średnimi oraz wysokimi częstotliwościami.

Strona celowo pozostawiona pusta.

2.12. Podsumowanie

Chociaż idea częstotliwościowej reprezentacji sygnału jest bardzo stara i sięga swoimi korzeniami już początków XIX wieku, to stanowi ona bardzo ważny element współczesnych metod analizy, przetwarzania oraz kompresji obrazów, i sygnałów w ogóle. Z tej perspektywy powszechnie dzisiaj stosowanym narzędziem analizy częstotliwościowej obrazów jest przekształcenie DCT, które z uwagi na swoje parametry (efektywność i relatywnie niewielka złożoność obliczeniowa) znajduje praktyczne wykorzystanie w znakomitej większości koderów obrazu. Ten ostatni fakt jest powodem, dla którego transformacji DCT została poświęcona tak szczególna uwaga w tym rozdziale.

Intencją tego rozdziału było również pokazanie, że istnieją także inne, obok transformacji DCT, sposoby reprezentacji obrazów, dla których istnieje wyraźne uzasadnienie ich wykorzystania w najnowszych koderach obrazu. W tym kontekście istotnym dopełnieniem DCT mogą być omówione wcześniej przekształcenia: WHT (zastosowanie w koderze obrazu dla celów szybkiego wyboru trybów kompresji fragmentów obrazu oraz wydajnej reprezentacji fragmentów kodowanych z użyciem wybranych trybów), oraz przekształcenia DST i KLT (wydajna kompresja danych wytwarzanych przez niektóre tryby kompresji bloków). Z uwagi na wskazane w tym rozdziale zalety przywołanych przekształceń (zalety WHT, DST, KLT w stosunku do DCT) coraz częściej stają się one elementem nowych technik kompresji obrazów. Jednakże z perspektywy właściwej kompresji danych obrazowych DCT ciągle pełni tutaj rolę dominującą.

Postęp w zakresie prędkości przetwarzania danych, ale również zwiększająca się rozdzielczość przestrzenna wyświetlanych obrazów istotnie zmieniły sposób, w jaki transformacja DCT jest stosowana w koderach obrazu. W starszych technikach kompresji (np. unormowana w 1991 roku technika JPEG kodowania statycznego obrazu) regułą było wyznaczanie próbek przekształcenia w blokach obrazu o z góry ustalonym, niewielkim rozmiarze (najczęściej był to blok 8x8 próbek). Nie było więc mechanizmu wyboru najlepszego rozmiaru bloku dla przekształcenia DCT, który byłby dopasowany do aktualnie kodowanej treści. W takim przypadku niektóre fragmenty obrazu (np. takie, w których treść zmienia się powoli) były kodowane mało wydajnie. Jednak w najnowszych koderach obrazu to się zmieniło, kiedy to stworzono koderowi możliwość wyboru rozmiaru bloku dla przekształcenia. Dobrym przykładem jest tutaj technika HEVC kodowania sekwencji wizyjnych. W tej technice próbki przekształcenia DCT mogą być wyznaczane w blokach o rozmiarze 4x4, 8x8, 16x16 oraz 32x32 próbki obrazu. Zależnie od charakteru aktualnie kodowanej treści koder może wybrać najlepszy wariant, co znacznie poprawia efektywność kompresji danych. Ponieważ rozwój multimediów idzie w kierunku dalszego zwiększenia rozdzielczości przestrzennej obrazów (czyli w ogólności jeszcze większa niż jest to obecnie, liczba próbek obrazu będzie wykazywać silne podobieństwo) to należy się spodziewać, że w przyszłych koderach obrazu możliwe będzie stosowanie DCT w jeszcze większych blokach.

Przedstawione w tym rozdziale sposoby (częstotliwościowej) reprezentacji danych obrazu faktycznie przeważają w aplikacjach rynkowych, przemysłowych. Należy sobie zdać jednak sprawę z tego, że nie są to wszystkie znane metody. Wspomnieć tutaj należy chociażby o **kodowaniu subpasmowym**, czy **falkowym** obrazów [Doma10], które również cieszyły się bardzo dużym zainteresowaniem wśród badaczy, szczególnie w latach 90-tych ubiegłego wieku i później (do roku 2003-2004). Według najlepszej wiedzy autora apogeum tego zainteresowania rozpoczęło się w roku 2000, kiedy to opracowana została technologia **JPEG 2000** kodowania statycznego obrazu [JPEG2000], do dziś wykorzystywana na przykład w kinie cyfrowym. Technologia ta pod względem efektywności kompresji biła wtedy na głowę inne znane techniki (również te, które

bazowały na DCT), i z tego właśnie powodu była przez wielu uważana jako najbardziej obiecujący kierunek badań w zakresie kompresji obrazów. Te przewidywania się jednak nie sprawdziły, głównie z uwagi na to, że niewątpliwy sukces JPEG 2000 nie został powtórzony w kontekście kodowania ruchomego obrazu. Właśnie rok 2004 (konkurs komitetu MPEG na technologię skalowalnej kompresji ruchomego obrazu) ostatecznie pokazał, że na tym polu bezkonkurencyjne są jednak **techniki hybrydowej kompresji** obrazów, których częścią jest oparte na DCT kodowanie transformatowe danych. Dlatego dostępnych koderów obrazu, które realizują koncepcję kompresji subpasmowej (czy falkowej) jest relatywnie niewiele. Stąd, w tym wydaniu książki, subpasmowa (czy falkowa) analiza i kompresja danych obrazu nie jest przedmiotem szerszej dyskusji.

Rozdział 3

Kodowanie entropijne

3.1. Wprowadzenie

Kodowanie entropijne jest techniką **kodowania bezstratnego** danych. Technika ta realizuje kodowanie symboli danych **z uwzględnieniem częstości występowania tych symboli w strumieniu danych.** Tak więc, przed właściwym etapem kodowania entropijnego każdemu symbolowi danych (pochodzącego ze zdefiniowanego wcześniej alfabetu $S = \{x_1, x_2, \dots, x_N\}$) przyporządkować należy prawdopodobieństwa $P = \{p(x_1), p(x_2), \dots, p(x_N)\}$ występowania poszczególnych symboli. Sposób wyznaczenia tych prawdopodobieństw ma olbrzymi wpływ na efektywność całej techniki kodowania entropijnego.

Główna idea kodowania entropijnego jest następująca. Każdemu symbolowi (lub ciągowi symboli) przypisuje się pewien ciąg bitów (słowo kodowe), uwzględniając statystykę kodowanego symbolu (lub ciągu symboli) w strumieniu danych. Aby takie kodowanie było efektywne (czyli prowadziło do redukcji ilości danych reprezentujących symbole danych) długość przypisywanego symbolom (lub ciągom symboli) słowa kodowego musi wynikać wprost z prawdopodobieństw ich występowania. I tak, symbole pojawiające się często (czyli te z dużym prawdopodobieństwem występowania) reprezentowane są krótszymi słowami kodowymi, w przeciwieństwie do symboli rzadko występujących (małe wartości prawdopodobieństwa występowania), które koduje się dłuższymi kodami. W ten sposób kodowanie entropijne dokonuje **redukcji statystycznej nadmiarowości (redundancji)**, jaka występuje w zbiorze kodowanych danych. Z oczywistych powodów nie można wszystkim symbolom danych przypisać krótkich kodów, bo te kody zaczęłyby się powtarzać dla różnych symboli i przez to stałyby się **niedekodowalne**.

Aby możliwe stało się bezstratne zakodowanie oraz jednoznaczne zdekodowanie w dekoderze wszystkich symboli alfabetu $S = \{x_1, x_2, \dots, x_N\}$, każdemu symbolowi danych należy przeznaczyć pewną liczbę bitów (ale nie mniejszą od pewnej minimalnej wartości), które muszą być przesłane do dekodera. W tym kontekście, najmniejsza możliwa długość słowa kodowanego danego symbolu x_k alfabetu $S = \{x_1, x_2, \dots, x_N\}$ (długość wyrażana w bitach, np. 1,234456 bita), która pozwala na bezbłędne przesłanie tego symbolu do dekodera zależy od prawdopodobieństwa $p(x_k)$ symbolu i zgodnie z **teorią Shannona** o bezstratnym kodowaniu źródła S [Shan48] wynosi:

$$\log_2\left(\frac{1}{p(x_k)}\right) \tag{3.1}$$

(**UWAGA:** Nie mówimy tutaj o sytuacji, w której dany symbol kodujemy na mniejszej niż wyrażonej wzorem 3.1 liczbie bitów kosztem wydłużenia słowa/słów kodowych innego/innych symboli alfabetu)

Mając alfabet $S = \{x_1, x_2, \dots, x_N\}$ składający się z N różnych symboli, jakie mogą wystąpić w strumieniu danych, oraz znając prawdopodobieństwa ich wystąpienia $P = \{p(x_1), p(x_2), \dots, p(x_N)\}$ wyznaczyć można **entropię**[25] źródła danych, która wyraża się wzorem 3.2.

$$H\{x_1, x_2, ..., x_N\} = \sum_{k=1}^{N} p(x_k) \cdot \log_2\left(\frac{1}{p(x_k)}\right) \tag{3.2}$$

W dziedzinie kompresji bezstratnej danych entropia źródła jest ważnym pojęciem, gdyż wyznacza ona **minimalną średnią długość słowa kodowego**, jaka przypada na jeden symbol danych, przy zapewnieniu jego bezstratnej kompresji i dekompresji. Z punktu widzenia efektywności kodowania entropijnego (wyrażanej średnią długością słowa kodowego symbolu danych) entropia źródła określa dolną granicę tej efektywności. Tym samym nie jest możliwe zaproponowanie kodu, dla którego średnia długość słowa kodowego będzie mniejsza od wartości wyrażonej entropią. Bardzo często, uzyskiwana w praktyce **średnia długość słowa kodowego** $\bar{L}$ symbolu (wyznaczona na podstawie faktycznej długości $l(x_k)$ słów kodowych symboli dla $k = 1,2, \dots, N$, oraz prawdopodobieństw $p(x_k)$ tych symboli – patrz wzór 3.3) jest większa od tej określonej entropią źródła (czyli $\bar{L} > H\{x_1, x_2, \dots, x_N\}$). Tylko w wyjątkowych przypadkach $\bar{L}$ jest równe entropii źródła (przykładowo, taka sytuacja wystąpi, jeśli wartości $\log_2\left(\frac{1}{p(x_k)}\right)$ dla wszystkich k będą całkowitoliczbowe). W ogólności zatem $\bar{L} \geq H\{x_1, x_2, \dots, x_N\}$.

$$\bar{L} = \sum_{k=1}^{N} l(x_k) \cdot p(x_k) \tag{3.3}$$

Relacja wartości $\bar{L}$ oraz $H\{x_1, x_2, \dots, x_N\}$ określa więc efektywność danej techniki kodowania entropijnego.

Na przestrzeni lat opracowano wiele różnych metod kodowania entropijnego. W kompresji danych, szczególne znaczenie zyskały techniki kodowania o zmiennej długości słowa kodowego (ang. Variable-Length Coding (VLC)), np. **kodowanie Huffmana** [Huff52, Sayo00], **kodowanie Golomba** [Golo66], **kodowanie Rice'a** [Rice79] (określane również jako kodowanie Golomba-Rice'a), czy **kodowanie Exp-Golomba** [Teuh78], techniki **kodowania arytmetycznego** [Riss79, Witt87, Said04], oraz metody **kodowania słownikowego** danych [Ziv77, Ziv78, Welch84]. Wymienione tutaj metody (chociaż metody kodowania słownikowego w dużo mniejszym stopniu) znajdują praktyczne zastosowanie w **kompresji stratnej i bezstratnej obrazów statycznych**, czy **stratnej kompresji cyfrowych sekwencji wizyjnych.** Jednak metody kodowania słownikowego nie są używane w zaawansowanych technikach kompresji obrazów i sekwencji wizyjnych. Wybrane metody kodowania VLC, jak również kodowanie arytmetyczne danych, które są szeroko stosowane w kompresji cyfrowego obrazu stanowią temat kolejnych punktów książki.

[25] Mowa jest tutaj tylko o entropii źródła zerowego rzędu. W literaturze przedmiotu znane są również pojęcia entropii źródła wyższych rzędów oraz entropii warunkowej.

3.2. Kodowanie o zmiennej długości słowa – VLC

Główna idea technik kodowania o zmiennej długości słowa kodowego (ang. Variable Length Coding (VLC)) jest bardzo prosta i została już wcześniej zasygnalizowana. W algorytmach tych, każdemu symbolowi alfabetu wejściowego przypisywany jest wprost, ściśle ustalony ciąg bitów (słowo kodowe), uwzględniając oczywiście częstość występowania tych symboli w kodowanym strumieniu danych. Aby uzyskać kompresję danych (czyli redukcję wielkości strumienia bitów reprezentującego dane) słowa kodowe o mniejszej długości przypisywane są częściej występującym symbolom danych (dla symboli rzadziej występujących zarezerwowane są kody dłuższe). Istnieje wiele różnych technik kodowania VLC, różniące się sposobem wyznaczania słów kodowych [Salom07]. Tylko część, z grupy znanych w literaturze metod VLC, znajduje zastosowanie w kompresji obrazu.

3.2.1. Kodowanie Huffmana

Kodowanie Huffmana jest jedną z najstarszych i zarazem jedną z najbardziej znanych metod kodowania entropijnego danych [Huff52, Przel05, Sayo00, Salom06, Salom07, Doma10]. Metoda ta charakteryzuje się **wysoką efektywnością kompresji** oraz **niewielką złożonością obliczeniową kodera i dekodera.** Stanowi to ogromną zaletę tej techniki kodowania. W związku z powyższym, kodowanie Huffmana, oraz modyfikacje tej metody, są po dzień dzisiejszy częścią wielu współczesnych systemów kompresji i archiwizacji danych, np. bzip2 [bzip2], czy gzip [gzip]. Na tym nie kończy się zastosowanie techniki kodowania Huffmana. Metoda ta jest również powszechnie wykorzystywana w kompresji danych multimedialnych: w międzynarodowych standardach kompresji dźwięku szerokopasmowego, np. MPEG-1 Layer 3 (MP3) [MPEG-1], MPEG-2 AAC [MP2AAC], MPEG-4 AAC [MP4AAC], w systemach kompresji obrazu statycznego, np. JPEG [JPEG], PNG [PNG], TIFF [TIFF], oraz kompresji cyfrowych sekwencji wizyjnych, np. MPEG-1 [MPEG-1], MPEG-2 [MPEG-2], H.263 [H263], VC-1 [VC-1], oraz MPEG-4 AVC/H.264 [AVC, Wieg03]. Kolejny punkt przedstawia sposób wyznaczania słów kodowych Huffmana w podstawowej wersji algorytmu.

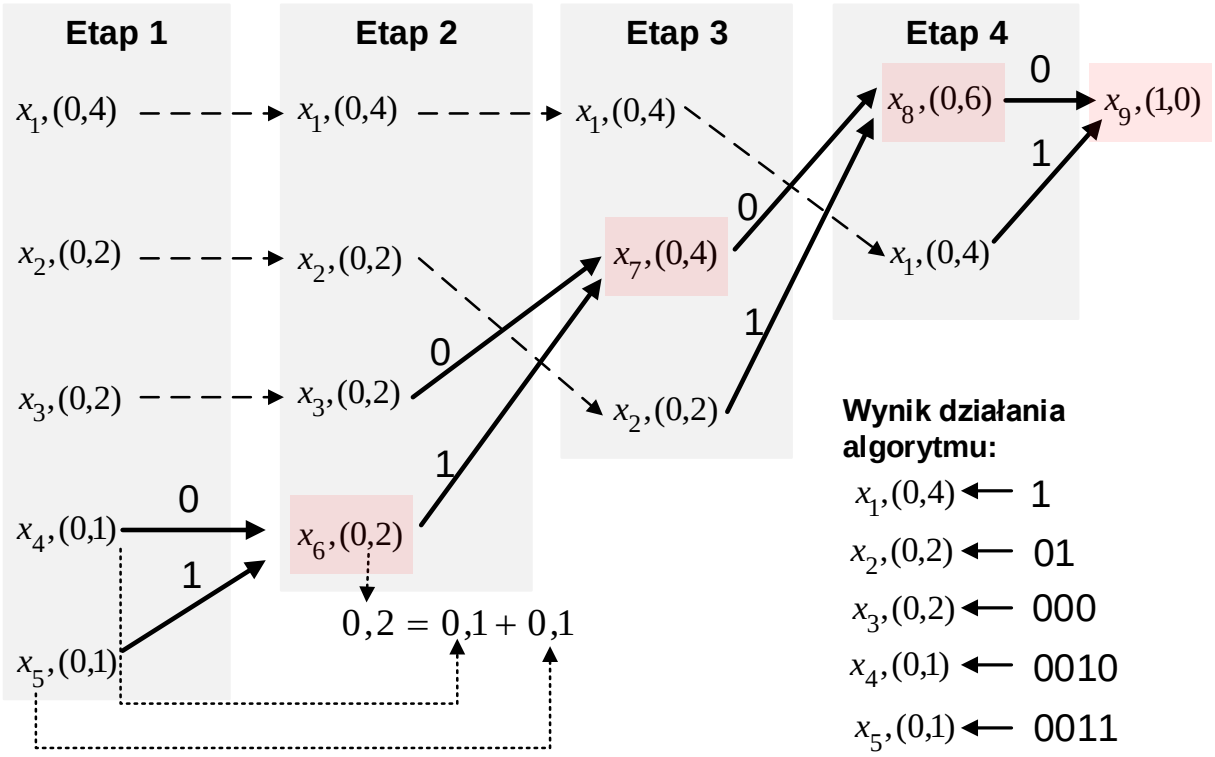
3.2.1.1. Kodowanie Huffmana – szczegóły algorytmu podstawowego

Wyznaczenie kodów Huffmana dla zbioru N symboli alfabetu $S = \{x_1, x_2, \dots, x_N\}$, które pojawiają się w strumieniu danych z prawdopodobieństwami odpowiednio $P = \{p(x_1), p(x_2), \dots, p(x_N)\}$ jest możliwe realizując kolejne etapy obliczeń (patrz Rysunek 3-1), z których każdy obejmuje następujące kroki:

Krok 1: Symbole alfabetu należy posortować zgodnie z prawdopodobieństwami występowania tych symboli w zbiorze kodowanych danych – w kolejności od największej do najmniejszej wartości prawdopodobieństwa.

Krok 2: Dwa najmniej prawdopodobne symbole łączone są w jeden nowy symbol, którego prawdopodobieństwo występowania jest sumą prawdopodobieństw łączonych symboli. Nowy symbol zastępuje symbole ‘źródłowe’, z których powstał. W ten sposób uzyskuje się zmieniony alfabet S symboli, którego liczność (z uwagi na dokonane połączenie dwóch symboli w jeden nowy symbol) jest o jeden mniejsza w stosunku do alfabetu z **Kroku 1**.

Wspomniane etapy obliczeń kontynuuje się do momentu uzyskania alfabetu zawierającego tylko jeden symbol. Prawdopodobieństwo tego symbolu wynosi 1, gdyż jest sumą prawdopodobieństw wszystkich symboli alfabetu S (patrz wynik etapu 4 w przykładzie z Rysunku 3-1).
Liczba omawianych etapów obliczeń wynika wprost z liczności początkowego alfabetu. Rysunek 3-1 ilustruje przedstawiony algorytm tworzenia kodów Huffmana dla przykładowego zbioru pięciu symboli danych $S = \{x_1, x_2, x_3, x_4, x_5\}$, pojawiających się z prawdopodobieństwami $P = \{0{,}4; 0{,}2; 0{,}2; 0{,}1; 0{,}1\}$.



Rysunek 3-1 Tworzenie kodów Huffmana dla przykładowego zbioru pięciu symboli danych. W nawiasach podano prawdopodobieństwa występowania symboli danych. Czerwonym oknem zaznaczono nowe symbole będące wynikiem połączenia symboli istniejących już wcześniej. Strzałki narysowane linią ciągłą reprezentują gałęzie wynikowego drzewa binarnego.

Wynikiem przedstawionego powyżej algorytmu jest **drzewo binarne** o korzeniu umiejscowionym w wynikowym symbolu, którego prawdopodobieństwo równe jest 1 (patrz wynik etapu 4 dla przykładu przedstawionego na Rysunku 3-1). Na poszczególnych głębokościach drzewa, górnej oraz dolnej gałęzi przypisuje się odpowiednio bity: **0** oraz **1** (możliwe jest również przypisanie odwrotne: **1** dla górnej gałęzi oraz **0** dla gałęzi dolnej). Wyznaczone dla symboli danych słowa kodowe otrzymuje się poprzez sczytanie bitów (**0** lub **1**) przypisanych gałęziom drzewa binarnego poruszając się po drzewie w kierunku od korzenia (ostatni etap obliczeń) do liści drzewa (wcześniejsze etapy obliczeń). Otrzymane w ten sposób kody Huffmana wykazują następujące cechy:

- Każdy symbol danych wejściowych reprezentowany jest przez słowo kodowe o pewnej całkowitoliczbowej długości.
- Dwa symbole o najniższym prawdopodobieństwie występowania w strumieniu danych posiadają słowa kodowe o tej samej długości (ale są to różne słowa kodowe).
- Żadne słowo kodowe nie jest początkiem (przedrostkiem) innego słowa kodowego (konieczny warunek z punktu widzenia dekodowalności kodu).

3.2.1.2. Efektywność podstawowej wersji kodowania Huffmana

Chociaż kodowanie Huffmana cechuje wysoka efektywność, to tylko w szczególnych przypadkach średnia długość $\bar{L}$ słowa kodowego równa jest granicznej wartości równej entropii $H\{x_1, x_2, \dots, x_N\}$ źródła. Mówimy wtedy, że kodowanie Huffmana jest optymalne i ma to miejsce tylko wtedy, gdy prawdopodobieństwa symboli danych są ujemnymi potęgami dwójki. Zwykle niestety tak nie jest i wtedy średnia długość $\bar{L}$ kodu Huffmana jest większa od granicznej wartości określonej entropią $H\{x_1, x_2, \dots, x_N\}$ źródła. W przedstawionym na rysunku 3-1 przykładzie otrzymano $\bar{L} = 2{,}2$ bita $> H = 2{,}121928$ bita. W ogólnym przypadku, uzyskiwana w kodowaniu Huffmana wartość $\bar{L}$ zawiera się w następujących granicach:

$$H\{x_1, x_2, \dots, x_N\} \leq \bar{L} \leq H\{x_1, x_2, \dots, x_N\} + 1 \tag{3.4}$$

Praca [Gall78] podaje w sposób dokładniejszy, niż ma to miejsce w przypadku powyższego wyrażenia, wartość górnej granicy dla uzyskiwanej w praktyce średniej długości $\bar{L}$ słowa kodowego Huffmana (dolna granica pozostaje niezmieniona: najlepszym możliwym wynikiem jest oczywiście $\bar{L} = H$). Jak się okazuje, wartość górnej granicy dla $\bar{L}$ zależy od wartości $p_{max} = max\{p(x_i)\}_{i=1}^{N}$, czyli wartości prawdopodobieństwa najbardziej prawdopodobnego symbolu danych. Górna granica określona jest jako:

$$\begin{aligned} p_{max} < 0{,}5 \quad &\Rightarrow \quad \bar{L} \leq H(S) + p_{max}, \\ p_{max} \geq 0{,}5 \quad &\Rightarrow \quad \bar{L} \leq H(S) + p_{max} + \sigma \end{aligned} \tag{3.5}$$

gdzie $\sigma = 1 - \log_2 e + \log_2(\log_2 e) \approx 0{,}086$.

Przedstawiona do tej pory **podstawowa wersja algorytmu Huffmana** generuje słowa kodowe, z których najkrótszy ma długość 1 bita. W tym kontekście, symbol danych x_k, którego prawdopodobieństwo występowania $p(x_k) > 0{,}5$, kodowany jest w sposób nieefektywny. Pośrednio widać to analizując wyrażenie 3.5. W tej sytuacji przeznaczamy 1-bitowe słowo kodowe dla symbolu, który zgodnie z wyrażeniem 3.1 można zakodować słowem o ułamkowej (mniejszej niż 1) liczbie bitów. Opisywana, niekorzystna sytuacja jest bardziej prawdopodobna w przypadku kodowania symboli danych pochodzących z alfabetu zawierającego niewielką liczbę elementów. Istnieje możliwość dodatkowego zwiększenia efektywności przedstawionego algorytmu Huffmana realizując tzw. **blokowe kodowanie Huffmana**.

3.2.2. Blokowe kodowanie Huffmana

Zamiast przypisywać słowa kodowe poszczególnym symbolom alfabetu S możliwe jest zgrupowanie dwóch, trzech lub większej liczby symboli danych w jeden nowy symbol (wyraz) i jemu następnie przypisać słowo kodowe. W ten sposób kodujemy bloki (ciągi) symboli danych w sposób łączny, a nie pojedyncze symbole alfabetu S niezależnie. Grupowanie symboli danych w bloki (ciągi) n-elementowe umożliwia zwiększenie efektywności kodowania danych, gdyż zwykle prowadzi to do zmniejszenia średniej długości słowa kodowego $\bar{L}$, jaka przypada na pojedynczy symbol alfabetu S. W przypadku grupowania oraz łącznego kodowania ciągu n symboli, średnia długość słowa kodowego $\bar{L}$ przypadająca na pojedynczy symbol oryginalnego alfabetu S zawiera się w następujących granicach [Sayo00]:

$$\boldsymbol{H\{x_1, x_2, \dots, x_N\} \leq \bar{L} \leq H\{x_1, x_2, \dots, x_N\} + \frac{1}{n}} \quad (3.6)$$

Oczywistym jest, że im większa długość n tworzonego ciągu symboli tym uzyskiwana efektywność kodowania jest bliższa granicznej wartości określonej entropią $H\{x_1, x_2, \dots, x_N\}$ źródła. W tym kontekście wejściowe symbole danych powinno się grupować w ciągi jak najdłuższe.

Grupowanie symboli danych w coraz dłuższe ciągi (bloki) wiąże się z innym istotnym problemem. Otóż szybko rośnie liczba wyrazów, którym należy przypisać słowa kodowe. Dla przykładu, jeśli oryginalny alfabet S składa się z N różnych symboli, to liczba różnych ciągów 2-elementowych (powstałych poprzez połączenie w pary symboli alfabetu S) wynosi już N^2. W przypadku ciągów 3-elementowych ich liczba to już N^3. I tak dalej. Tak więc, w stosunku do rozmiaru oryginalnego alfabetu S łączenie symboli w bloki (ciągi) skutkuje wykładniczym wzrostem liczby nowych elementów, a tym samym, wykładniczym wzrostem liczby kodów Huffmana przypisanych do tych elementów. Z uwagi na konieczność przechowywania kodów Huffmana w pamięci kodera i dekodera, wykładniczy wzrost liczby tych kodów silnie zwiększa pamięciową złożoność algorytmu kodowania i dekodowania. Poza wzrostem złożoności pamięciowej rośnie również obliczeniowa złożoność kodowania i dekodowania. Zwiększona liczba kodów Huffmana wydłuża czas tworzenia słów kodowych po stronie kodera. Dekodowanie słów kodowych Huffmana pochodzących z licznego zbioru kodów też zwykle wymaga większej ilości obliczeń procesora. Stanowi to istotne ograniczenie techniki blokowego kodowania Huffmana, szczególnie w przypadkach kodowania danych pochodzących z alfabetu o dużym rozmiarze (dużej liczbie symboli alfabetu).

3.2.3. Efektywność przedstawionych technik kodowania Huffmana w praktyce kompresji danych multimedialnych – krótki komentarz

W obliczu z góry znanej statystyki kodowanych danych kodowanie Huffmana wykazuje bardzo wysoką efektywność. Granice tej efektywności (dolna oraz górna) zostały matematycznie wyrażone, co przedstawiono w poprzednich punktach książki. Realizując w praktyce entropijne kodowanie danych nie zawsze mamy jednak szczegółową wiedzę o częstości występowania poszczególnych symboli w strumieniu danych. Po stronie **kodera** danych możliwe jest zastosowanie tzw. dwuetapowego kodowania symboli, w którym etap drugi realizujący właściwą kompresję danych poprzedzony jest etapem pierwszym, w którym koder „uczy się” statystyki danych, które ma zakodować. Takie podejście, chociaż obliczeniowo bardziej kosztowne, pozwala na lepszy dobór słów kodowych, zwiększając tym samym efektywność kodowania entropijnego. Problemem jest jednak to, że wyliczona w koderze statystyka danych nie jest z góry znana w **dekoderze**. Dlatego też, koder musi poinformować dekoder o statystyce danych, które będą w dekoderze dekodowane. W realizowanym w ten sposób kodowaniu Huffmana, koder wysyła do dekodera dwa strumienie danych: strumień zawierający zakodowane symbole, oraz dodatkowo, strumień zawierający wyznaczoną tablicę kodów Huffmana. W ten sposób, konieczność przesłania do dekodera tablicy dedykowanych kodów (oprócz właściwego strumienia zakodowanych symboli) tak mocno „psuje” uzyskaną wcześniej wysoką efektywność kompresji, że rozwiązanie takie bardzo rzadko stosuje się w praktyce. Należy tutaj jeszcze raz mocno podkreślić, że przesłanie do dekodera tablicy kodów jest w omawianym scenariuszu kodowania konieczne, celem zapewnienia dekodowalności zakodowanego strumienia danych w dekoderze.

W praktyce kompresji danych multimedialnych (obraz i dźwięk) najczęściej stosuje się rozwiązanie, w którym w koderze i dekoderze na stałe zaszyta jest pewna „uniwersalna” tablica kodów Huffmana, przez co nie jest ona w ogóle przesyłania z kodera do dekodera (do dekodera

wysyłany jest tylko strumień zakodowanych symboli danych). Zdecydowana większość współczesnych technik kompresji danych multimedialnych realizuje takie właśnie rozwiązanie (np. MPEG-2, MPEG-4 AVC/H.264, MPEG-1 Layer 3 (MP3), MPEG-4 AAC). Podstawą dla wyznaczenia „uniwersalnej” tablicy kodów Huffmana jest pewien zbiór reprezentatywnych danych testowych, które w sposób wystarczająco dokładny będą odzwierciedlać charakter (statystykę) danych poddawanych w koderze kompresji. Rozwiązanie takie daje najczęściej bardzo dobre rezultaty. Zdarzają się jednak przypadki, w których statystyka kodowanych danych odbiega od charakteru danych testowych (które stanowiły punkt odniesienia dla opracowania tablicy kodów), i w efekcie efektywność kompresji wyraźnie spada. Tak więc, sposób określenia danych uczących (testowych) oraz stopień ich zgodności z danymi poddawanymi później kompresji w koderze determinują efektywność omawianego scenariusza kodowania danych.

Istnieją również przypadki, w których charakter danych poddawanych kompresji jest trudny do przewidzenia i/lub mocno zależy od wybranych parametrów kodowania (np. szerokość kroku kwantyzacji w przypadku metod kodowania stratnego). W takich sytuacjach przedstawione powyżej rozwiązanie z „uniwersalną” tablicą słów kodowych może być nieefektywne. Efektywność kompresji będzie tym niższa, im statystyka kodowanych danych będzie bardziej odbiegać od tej założonej, wyrażonej statystyką danych testowych (danych uczących). Rozwiązaniem tego problemu może być zastosowanie w koderze i dekoderze wielu wyznaczonych wcześniej „uniwersalnych” tablic słów kodowych (a nie tylko jednej) i przełączanie się pomiędzy tymi tablicami w trakcie kodowania oraz dekodowania danych, uwzględniając charakter (typ) przetwarzanych danych. Rozwiązanie z wieloma „uniwersalnymi” tablicami pozwala w sposób istotny zwiększyć efektywność kompresji danych, jednak wzrasta w tej sytuacji pamięciowy koszt zapamiętania tablic kodów w koderze i dekoderze.

Alternatywnym rozwiązaniem, w stosunku do przedstawionych wyżej, jest **dynamiczne kodowanie Huffmana**, będące tematem kolejnego punktu.

3.2.4. Dynamiczne kodowanie Huffmana

3.2.4.1. Opis ogólny

W przeciwieństwie do omówionych wcześniej technik kodowania Huffmana (algorytm podstawowy oraz blokowe kodowanie Huffmana) w **dynamicznym kodowaniu Huffmana** nie jest tworzona jedna (ostateczna) wersja tablicy słów kodowych Huffmana (drzewo kodów Huffmana) dla danych źródła. W tym przypadku drzewo kodów Huffmana jest budowane oraz uaktualniane na bieżąco, w trakcie kodowania/dekodowania kolejnych symboli danych. Tworzone są w ten sposób kolejne wersje drzewa kodów Huffmana wyznaczone dla zbioru już zakodowanych/zdekodowanych danych. Należy tutaj podkreślić, że wraz z pojawieniem się nowego symbolu danych koder (jak również dekoder) nie tworzą nowego drzewa kodów od „zera” (nawiasem mówiąc takie rozwiązanie byłoby obliczeniowo bardzo złożone). Zamiast tego, koder i dekoder modyfikują aktualną wersję drzewa kodów według ściśle określonej procedury uaktualnienia drzewa (patrz punkt 3.2.4.3).

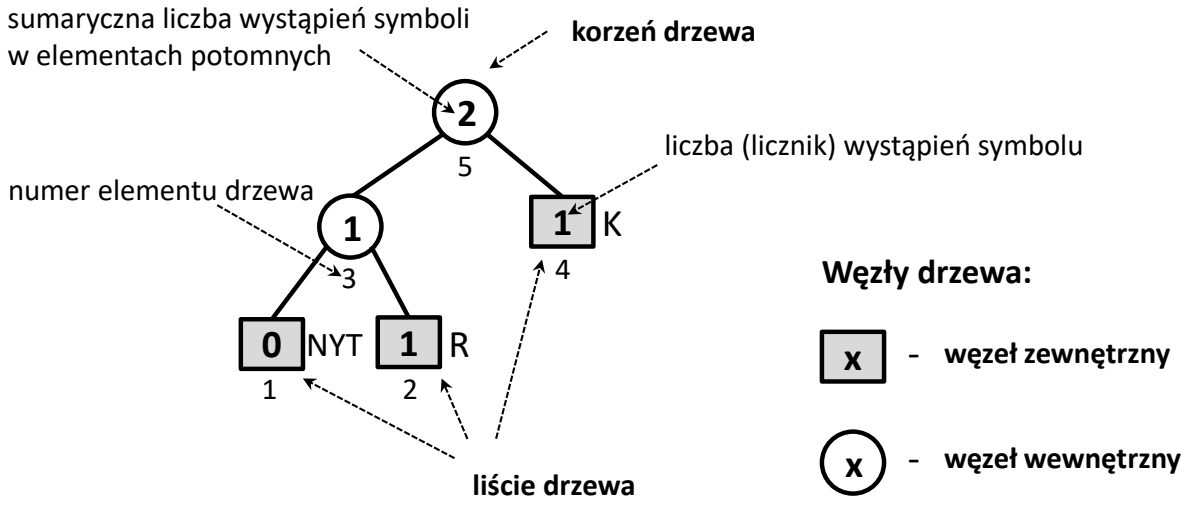
Koder (dekoder) rozpoczyna pracę z pustym drzewem Huffmana. Wraz z pojawieniem się nowego symbolu danych algorytm sprawdza, czy taki symbol został już wcześniej do drzewa dodany. Jeśli symbol nie znajduje się na drzewie, należy poinformować o tym dekoder. W tym przypadku do dekodera wysyłany jest zdefiniowany wcześniej kod **NYT** (ang. **N**ot **Y**et **T**ransmitted), po czym koduje się aktualny symbol na stałej, ustalonej wcześniej liczbie bitów. Po zakodowaniu symbolu jest on dodawany do drzewa i zostaje mu przydzielony kod Huffmana. Jeśli

nowy symbol wystąpił już wcześniej (czyli znajduje się już na drzewie Huffmana) to został mu już wcześniej przydzielony kod Huffmana – wysłanie tego kodu koduje symbol. Po zakodowaniu symbolu należy sprawdzić, czy nie zachodzi konieczność uaktualnienia (reorganizacji) drzewa Huffmana (patrz punkt 3.2.4.3). W przypadku uaktualnienia drzewa, zmieniają się słowa kodowe przypisane do poszczególnych symboli alfabetu danych.

Żeby otrzymany strumień zakodowanych danych był dekodowalny w dekoderze, ważne jest, aby każdorazowe uaktualnienie drzewa kodów było realizowane w oparciu o już przetworzone dane, które znane są zarówno w koderze jak i w dekoderze. Po spełnieniu tego warunku, działanie dekodera stanowi „lustrzane odbicie” kroków realizowanych w koderze.

3.2.4.2. Opis szczegółowy – drzewo kodów Huffmana

Z puntu widzenia efektywności techniki dynamicznego kodowania Huffmana kluczowy jest sposób uaktualnienia (reorganizacji) drzewa kodów po zakodowaniu/zdekodowaniu nowego symbolu danych. W tym celu w węzłach drzewa skojarzonych z poszczególnymi symbolami danych (tzw. liście drzewa, węzły zewnętrzne) należy przechowywać informację o częstości występowania poszczególnych symboli w zbiorze danych zakodowanych/zdekodowanych do tej pory. Dodatkowo, w węzłach (wewnętrznych) drzewa będących „rodzicami” węzłów „potomnych” przechowywana jest skumulowana informacja o częstości wystąpień symboli, będąca sumą odpowiednich liczników z węzłów potomnych znajdujących się w drzewie o jeden poziom niżej. Dla lepszego zrozumienia procedury uaktualnienia drzewa kodów Huffmana, węzły drzewa zostaną również ponumerowane przechodząc po drzewie od liści drzewa do jego korzenia, w kierunku od lewej do prawej na każdym poziomie węzłów w drzewie. Przyjęta konwencja opisu drzewa kodów Huffmana została przedstawiona na poniższym rysunku.



Rysunek 3-2 Przyjęty w pracy sposób prezentacji drzewa kodów w dynamicznym kodowaniu Huffmana. Pomysł prezentacji drzewa za książkami [Sayo00, Sayo12].

3.2.4.3. Opis szczegółowy – uaktualnienie drzewa kodów Huffmana

W procesie kodowania/dekodowania kolejnych symboli danych konieczne jest sprawdzenie, czy aktualne drzewo kodów jest optymalnym zbiorem kodów Huffmana dla zbioru

już zakodowanych/zdekodowanych symboli danych. Takiego sprawdzenia należy dokonać w oparciu o wiedzę o częstościach wystąpienia symboli danych w zakodowanym już zbiorze. Przeglądając kolejne poziomy węzłów w drzewie kodów (w kolejności od lewej do prawej na danym poziomie, oraz od liści drzewa do jego korzenia) sczytuje się liczniki wystąpień symboli zapisane w węzłach drzewa. Jeśli otrzymany w ten sposób ciąg liczb jest niemalejący, mówi się, że węzły znajdują się na odpowiednich miejscach w drzewie, a badane drzewo jest drzewem kodów Huffmana. W przeciwnym razie drzewo wymaga uaktualnienia (reorganizacji) celem otrzymania zbioru optymalnych kodów. Uaktualniając drzewo kodów kierujemy się zasadą, zgodnie z którą, symbole występujące częściej powinny się znaleźć bliżej korzenia drzewa (czyli w efekcie mieć przypisane krótsze słowa kodowe), w stosunku do symboli o mniejszym prawdopodobieństwie wystąpienia (czyli z dłuższymi słowami kodowymi).

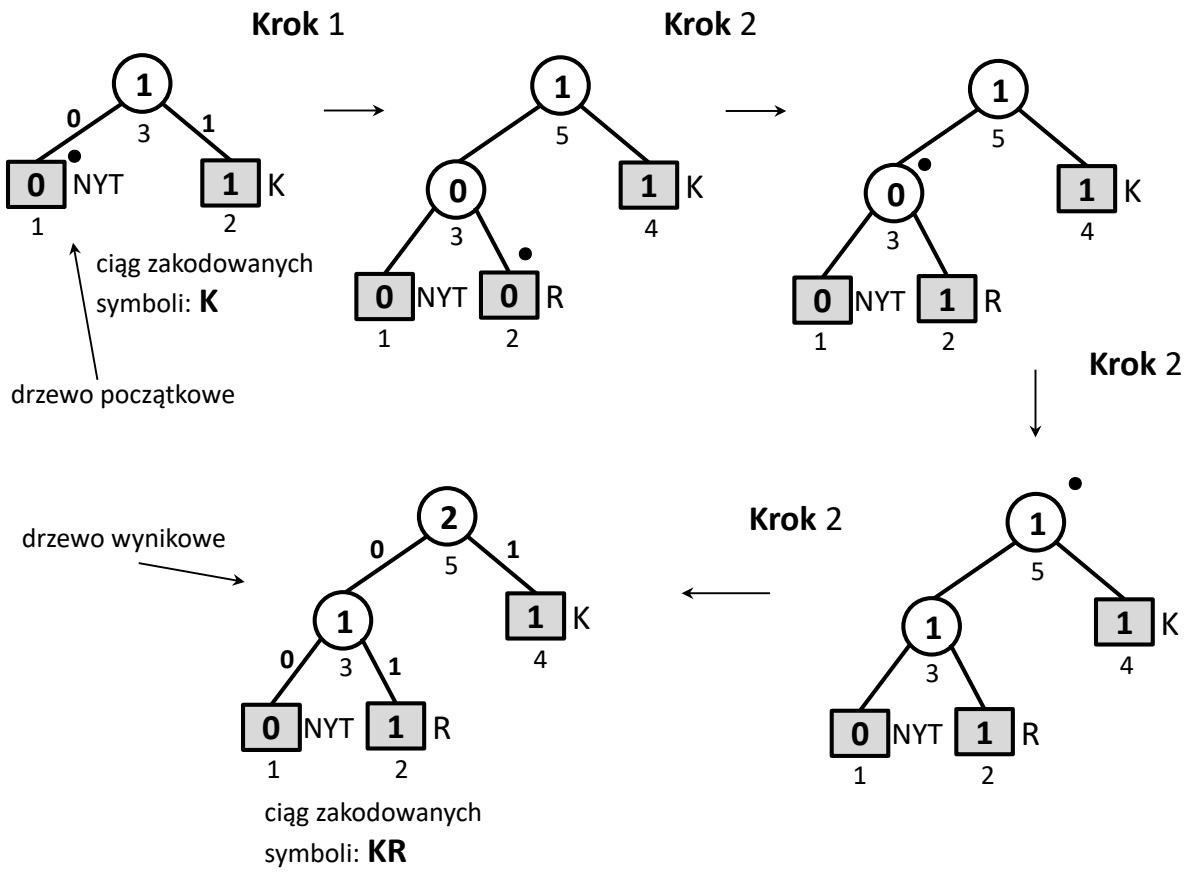
Przedstawioną powyżej koncepcję modyfikacji drzewa kodów realizuje się po zakodowaniu/zdekodowaniu aktualnego symbolu danych z wykorzystaniem przedstawionych poniżej szczegółowych algorytmów.

Algorytm 1 – zakodowano nowy symbol, który nie pojawił się nigdy wcześniej:

Krok 1: Nowy symbol dodawany jest do drzewa. W tym celu, zewnętrzny węzeł skojarzony dotychczas z symbolem **NYT** staje się węzłem wewnętrznym oraz rodzicem dwóch nowych węzłów potomnych: nowego węzła **NYT** oraz węzła skojarzonego z nowym symbolem. Liczniki częstości wystąpień symboli NYT oraz ostatnio dodanego do drzewa są ustawiane na 0.

Krok 2: Licznik częstości wystąpień nowego symbolu zwiększany jest o 1. Tym samym zachodzi konieczność aktualizacji liczników w węzłach „rodzicach” znajdujących się na wyższych poziomach w drzewie, aż do osiągnięcia korzenia drzewa.

Przedstawione kroki algorytmu zostały dodatkowo zilustrowane rysunkiem 3-3.



Rysunek 3-3 Kodowanie nowego symbolu **R** – ilustracja procedury dodawania nowego symbolu do drzewa Huffmana. Węzeł drzewa, dla którego dokonuje się zmian został oznaczony kropką (●).

Algorytm 2 – zakodowano symbol, który już wcześniej wystąpił:

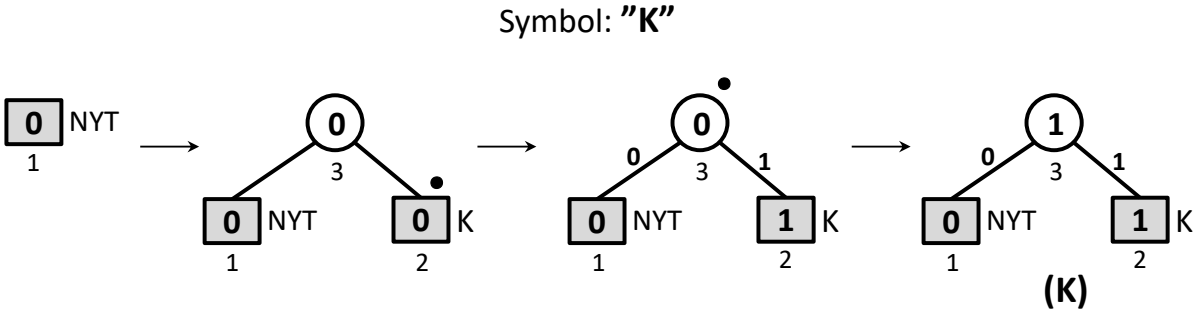
Krok 1: Odnajdujemy na drzewie węzeł skojarzony z zakodowanym symbolem. Węzeł ten staje się węzłem aktualnie przetwarzanym. Pobieramy z tego węzła informację o tym, ile razy zakodowany symbol pojawił się wcześniej w strumieniu danych.

Krok 2: Począwszy od węzła aktualnie przetwarzanego poruszamy się po drzewie wzdłuż ścieżki biegnącej w kierunku od bieżącego poziomu w drzewie do jego korzenia, oraz od strony lewej do strony prawej na każdym z poziomów drzewa. Odwiedzając w ten sposób wybrane węzły drzewa sprawdzamy zapisane w węzłach liczniki częstości wystąpień symboli danych. Porównując pomiędzy sobą wartości liczników sprawdzamy, czy istnieje możliwość przesunięcia węzła aktualnie przetwarzanego w górę ścieżki[26] (**Komentarz:** Węzły ścieżki są uszeregowane zgodnie z niemalejącymi wartościami liczników częstości wystąpień symboli. Na ścieżce może znajdować się grupa węzłów, których wartość licznika częstości wystąpień symboli jest taka sama jak w węźle aktualnie przetwarzanym (przykładowa grupa węzłów ścieżki została zilustrowana na rysunku 3-9). W tej sytuacji przesuwamy aktualnie przetwarzany węzeł (oraz jego poddrzewo) na koniec tej grupy (chodzi o to, żeby aktualnie przetwarzany węzeł miał najwyższy numer w grupie). Przesunięcia tego dokonujemy zamieniając miejscami aktualnie przetwarzany węzeł z odpowiednim węzłem grupy. **Uwaga, wyjątek**: Nie zamieniamy miejscami węzła aktualnie przetwarzanego z jego „rodzicem”.)

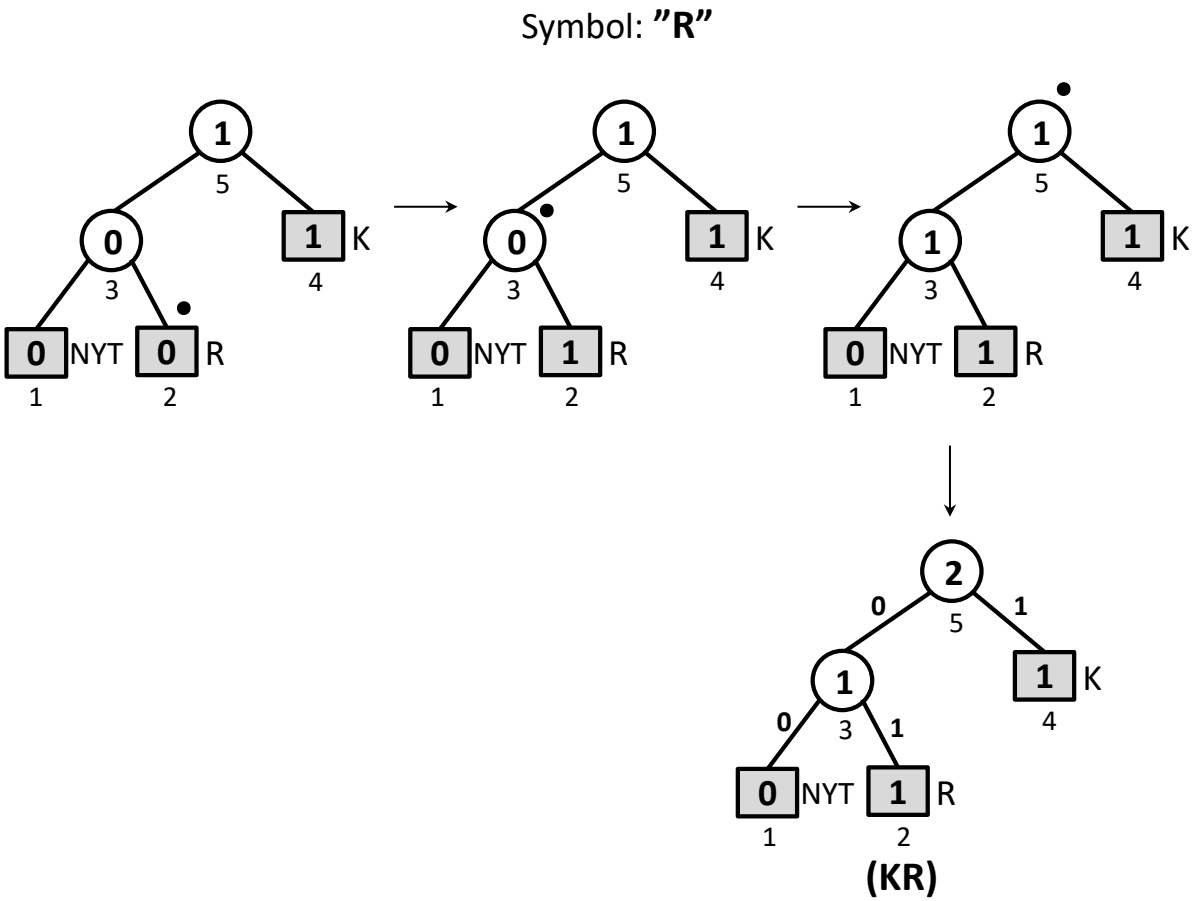
[26] Im bliżej korzenia drzewa znajduje się węzeł, tym krótszy kod Huffmana jest mu przypisany. Dlatego sprawdzamy czy istnieją przesłanki do przesunięcia zakodowanego symbolu w górę drzewa, czyli bliżej jego korzenia.

Krok 3: Zwiększamy o 1 licznik częstości wystąpień symbolu w aktualnie przetwarzanym węźle i przechodzimy do „rodzica” tego węzła. Przechodzimy do Kroku 2, w którym węzeł „rodzic” staje się aktualnie przetwarzanym węzłem. Powtarzamy całą procedurę aż do osiągnięcia korzenia drzewa.

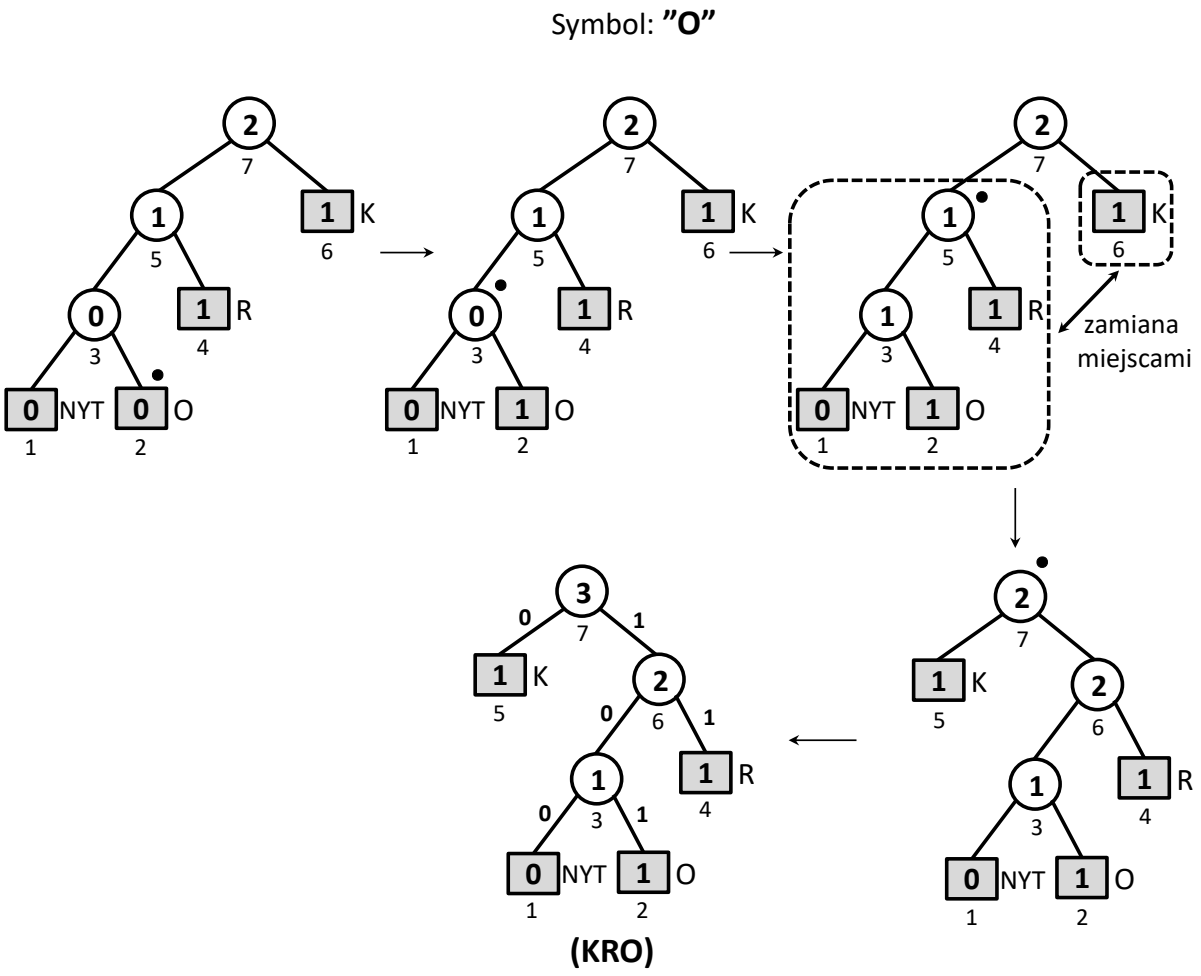
Celem lepszego zrozumienia przedstawionych algorytmów autor opracował przykład kodowania sekwencji symboli: **KROKUS**. Na zamieszczonych poniżej rysunkach zilustrowano poszczególne etapy uaktualnienia drzewa Huffmana, po zakodowaniu kolejnych symboli sekwencji.



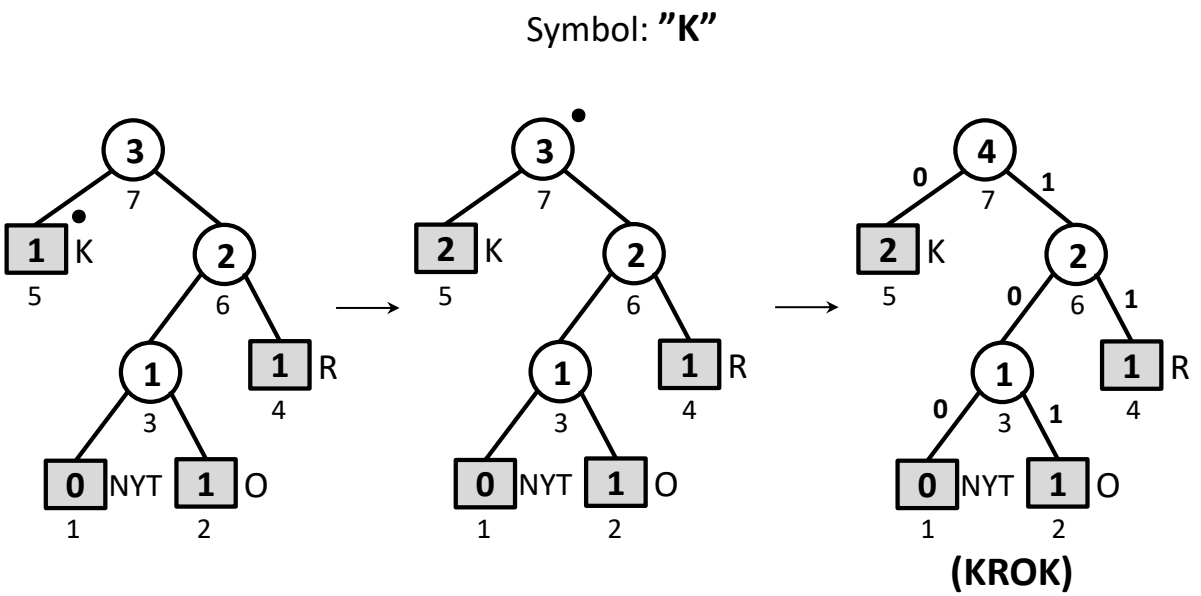
Rysunek 3-4 Kodowanie symbolu **K** – ilustracja uaktualnienia drzewa Huffmana. Aktualnie przetwarzany węzeł został oznaczony kropką (●).



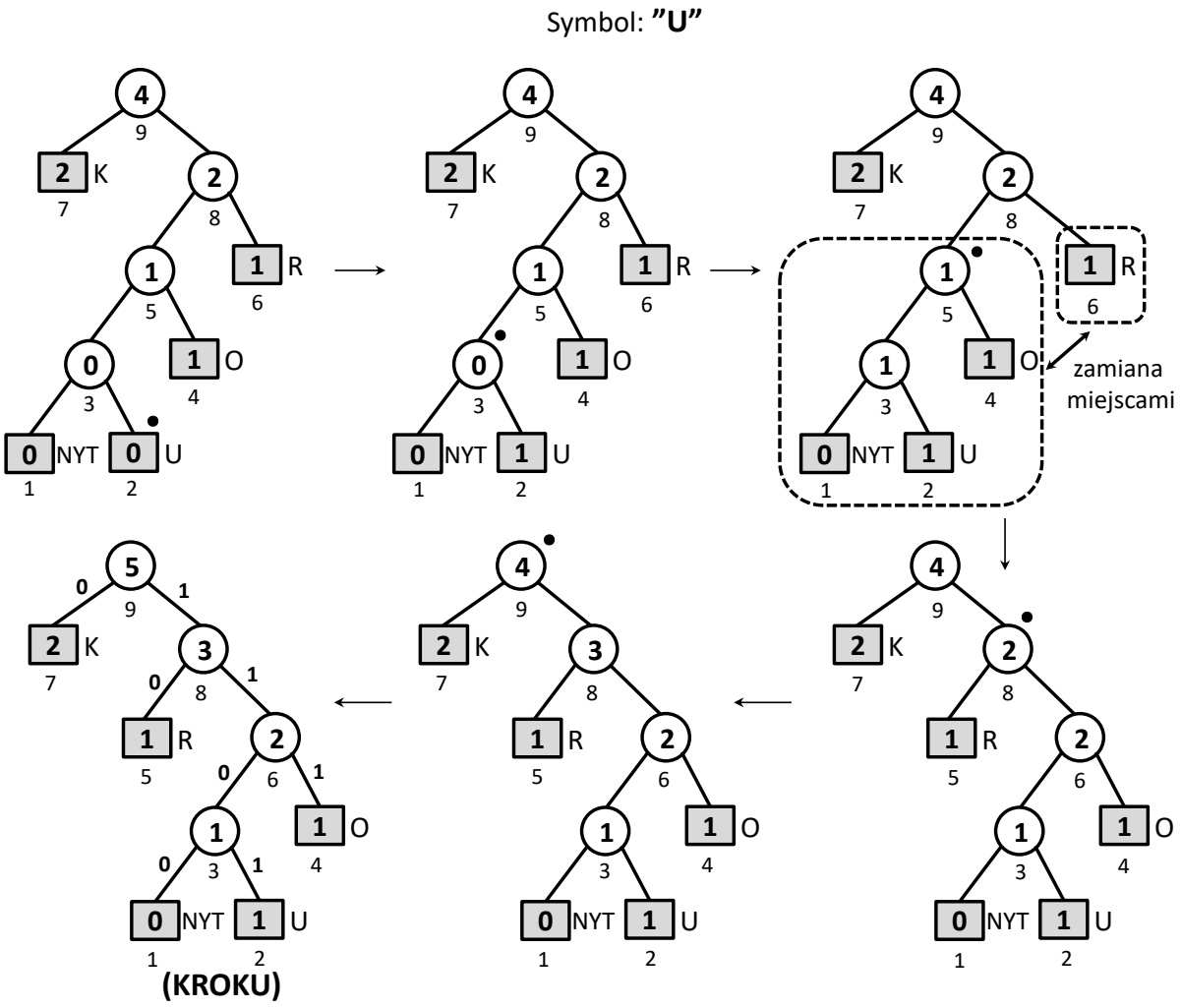
Rysunek 3-5 Kodowanie symbolu **R** – ilustracja uaktualnienia drzewa Huffmana. Aktualnie przetwarzany węzeł został oznaczony kropką (●).



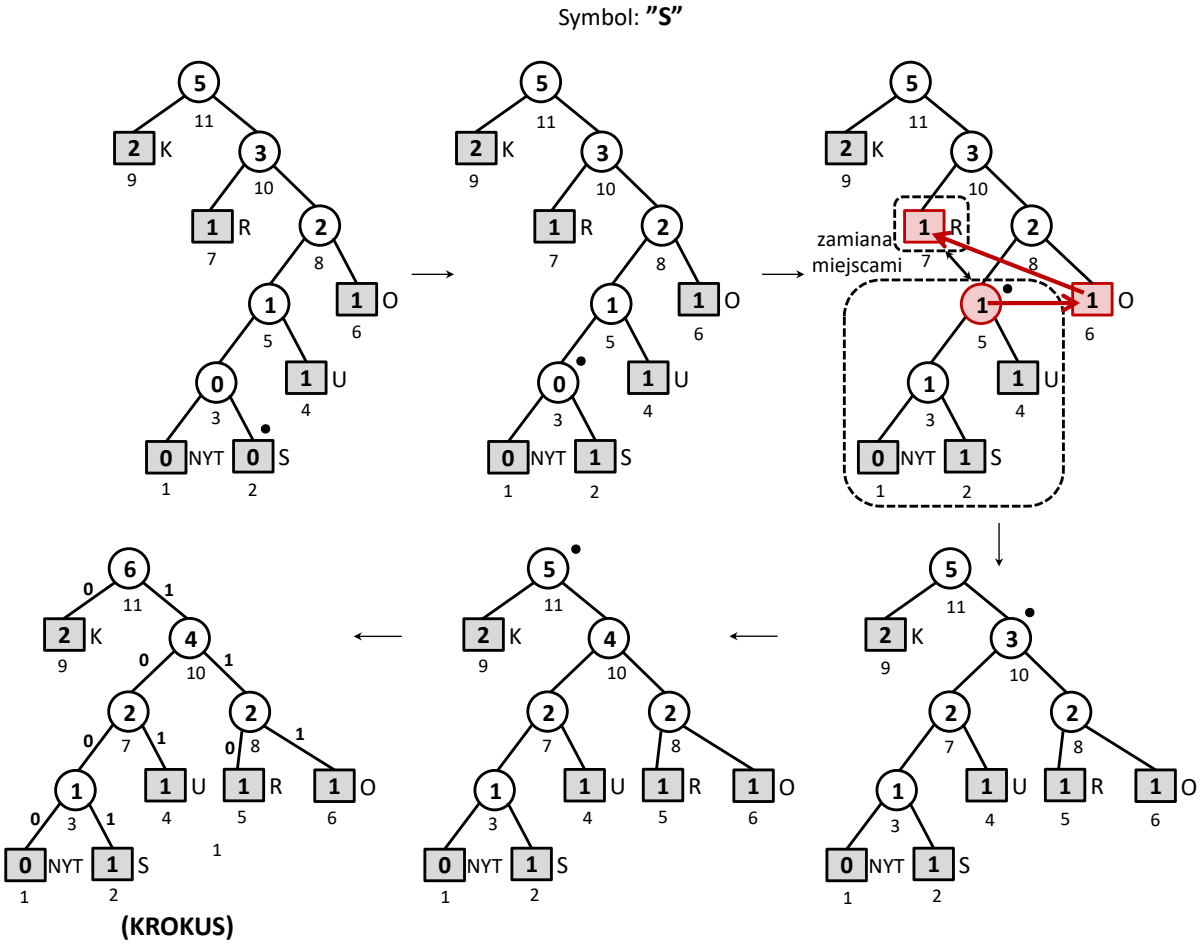
Rysunek 3-6 Kodowanie symbolu **O** – ilustracja uaktualnienia drzewa Huffmana. Aktualnie przetwarzany węzeł został oznaczony kropką (●).



Rysunek 3-7 Kodowanie symbolu **K** – ilustracja uaktualnienia drzewa Huffmana. Aktualnie przetwarzany węzeł został oznaczony kropką (●).



Rysunek 3-8 Kodowanie symbolu **U** – ilustracja uaktualnienia drzewa Huffmana. Aktualnie przetwarzany węzeł został oznaczony kropką (●).



Rysunek 3-9 Kodowanie symbolu **S** – ilustracja uaktualnienia drzewa Huffmana. Aktualnie przetwarzany węzeł został oznaczony kropką (●). Kolorem czerwonym zaznaczono na jednym z rysunków ścieżkę na drzewie, która ma swój początek w węźle aktualnie przetwarzanym.

3.2.5. „Uniwersalne” kodowanie o zmiennej długości słowa kodowego

Przedstawione w poprzednich punktach techniki kodowania Huffmana wymagają przechowywania wyznaczonych słów kodowych w pamięci kodera oraz dekodera. Jeśli z użyciem podstawowego algorytmu Huffmana kodujemy dane pochodzące z alfabetu o dużym rozmiarze, ale również w przypadku blokowego kodowania Huffmana, przechowywanie w pamięci kodera i dekodera słów kodowych Huffmana może być bardzo kosztowne.

Istnieje możliwość zmniejszenia wymagań pamięciowych w opisywanych przypadkach stosując tzw. **„uniwersalne” techniki kodowania entropijnego**. Cechą charakterystyczną tych technik jest prosty, z góry określony algorytm konstrukcji słów kodowych, dzięki któremu nie ma potrzeby budowania książki kodowej i jej przechowywania w pamięci urządzenia kodującego oraz dekodującego. Dodatkowo, złożoność obliczeniowa kodowania i dekodowania kodu „uniwersalnego” jest zwykle mniejsza od złożoności innych technik kodowania entropijnego (np. kodowania Huffmana czy kodowania arytmetycznego). Techniki „uniwersalnego” kodowania entropijnego, oprócz wymienionych zalet, mają również wadę: nie są technikami ogólnego przeznaczenia. Po pierwsze, techniki te stosuje się dla danych będących liczbami całkowitymi. Po drugie, postać słowa kodowego (oraz jego długość) wynika wprost z wartości kodowanej liczby.

Powoduje to, że techniki te wykazują wysoką efektywność kompresji tylko w przypadku kodowania wartości liczbowych o ściśle określonym rozkładzie statystycznym. Istnieje oczywiście możliwość zastosowania metod kodowania „uniwersalnego” dla danych innego typu niż liczby o wartości całkowitej, np. dla danych będących literami, jednak w tym przypadku należy dokonać odpowiedniego przyporządkowania kodowanych liter do liczb, by następnie kodować już liczby.

Znanych jest wiele technik kodowania „uniwersalnego”, których dokładny opis znaleźć można w literaturze [Salom06, Salom07, Salom10]. Do technik najbardziej popularnych zaliczyć można: **kodowanie Eliasa** [Elias75], **kodowanie Golomba** [Golo66], **kodowanie Fibonacciego** [Apos85], **kodowanie unarne** [Sayo00], **kodowanie Rice’a** (znane również jako kodowanie Golomba-Rice’a) [Rice79]. Niektóre z wymienionych technik znalazły praktyczne zastosowanie w kompresji danych multimedialnych, np.:

- kodowanie Golomba w standardzie JPEG-LS kompresji obrazu nieruchomego [JPEGLS],
- kodowanie Rice’a w technice FLAC bezstratnej kompresji dźwięku [Salom10],
- kodowanie unarne w standardach MPEG-4 AVC/H.264 oraz HEVC kompresji cyfrowych sekwencji wizyjnych [AVC, Richa03, Richa10, HEVC],
- kodowanie Exp-Golomba *k*-tego rzędu w standardach MPEG-4 AVC/H.264, AVS, HEVC [AVC, AVS, HEVC],
- kodowanie Golomba-Rice’a w standardzie HEVC kompresji ruchomego obrazu.

Główne założenia wybranych technik kodowania „uniwersalnego” przedstawiono w dalszej części tekstu.

3.2.5.1. Kodowanie unarne

Kodowanie unarne [Richa03, Przel05, Salom06, Salom07, Salom10] jest jedną z najprostszych metod kodowania „uniwersalnego”. Dla danej całkowitoliczbowej wartości $n \geq 0$ kod unarny składa się z n bitów o wartości 1 zakończonych bitem o wartości 0 (lub alternatywnie: n bitów o wartości 0 zakończonych bitem o wartości 1). Dla przykładu, kod 1110 odpowiada wartości całkowitej $n = 3$ natomiast kod 1111111110 reprezentuje wartość całkowitą $n = 9$. Taka postać kodu unarnego powoduje, że jest on efektywny w przypadku kodowania wartości liczbowych n pojawiających się w strumieniu danych z prawdopodobieństwem:

$$\boldsymbol{p(n) \cong \frac{1}{2^n}} \tag{3.7}$$

W przypadku kodowania wartości $n > 0$ (brak wartości zerowej w zbiorze kodowanych wartości) kod unarny może zostać uproszczony w taki sposób, aby składał się z $n - 1$ bitów o wartości 1 zakończonych bitem o wartości 0 (lub alternatywnie: $n - 1$ bitów o wartości 0 zakończonych bitem o wartości 1). W tym przypadku, kod 110 odpowiada wartości $n = 3$.

Olbrzymią zaletą kodowania unarnego jest niska złożoność kodowania wartości i dekodowania kodu, co wynika wprost ze struktury kodu i sposobu jego tworzenia. Utworzenie po stronie kodera kodu dla wartości $n \geq 0$ sprowadza się do odpowiedniego złożenia operacji przesunięć bitowych z operacją odjęcia wartości 1 (dla wartości n kod unarny można utworzyć realizując działania: $((1 \ll n) - 1) \ll 1$, gdzie ‘≪’ jest operacją dokonującą przesunięcia w lewo bitów reprezentacji binarnej liczby). Dekodowanie kodu sprowadza się natomiast do zliczenia liczby ‘jedynek’ w kodzie. W przypadku procesorów z rodziny x86 można to bardzo wydajnie zrealizować wykorzystując instrukcję BSR (Bit Scan Reverse) procesora. Jak widać, złożoność

obliczeniowa kodowania/dekodowania unarnego jest dużo mniejsza od obliczeniowej złożoności kodowania/dekodowania Huffmana.

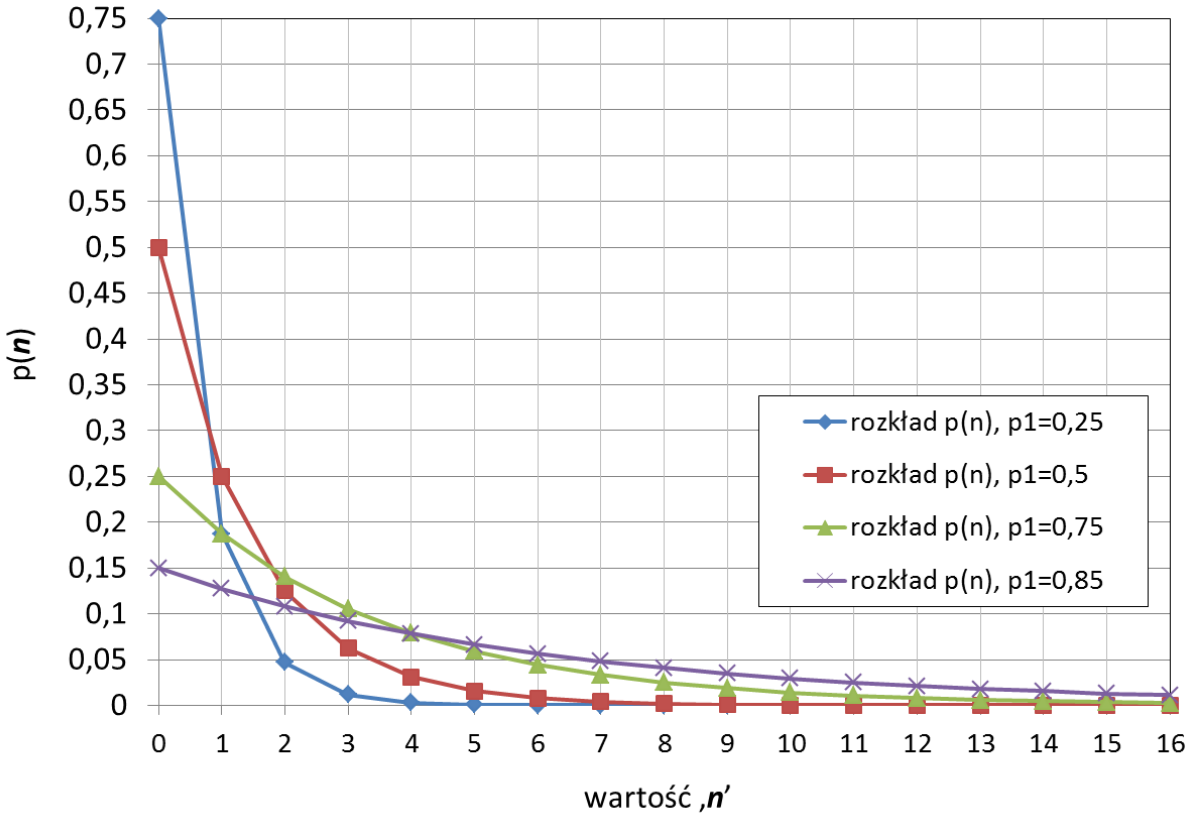
3.2.5.2. Kodowanie Golomba

Kody Golomba zostały opracowane z myślą o efektywnej reprezentacji powtórzeń tego samego symbolu danych [Golo66]. Zakłada się tutaj ściśle określony rozkład prawdopodobieństwa krotności powtórzenia kodowanego symbolu. I tak, kody Golomba zostały zaprojektowane z uwzględnieniem następujących założeń. Prawdopodobieństwo jednokrotnego wystąpienia kodowanego symbolu wynosi p_1. Prawdopodobieństwo ciągu będącego dwukrotnym powtórzeniem kodowanego symbolu jest mniejsze i wynosi $p_1 \cdot p_1 = p_1^2$. W ogólności, ciąg będący n-krotnym powtórzeniem symbolu pojawia się w strumieniu danych z prawdopodobieństwem p_1^n. Jeśli uwzględni się dodatkowo fakt, że omawiany ciąg n symboli zostanie zakończony symbolem o innej wartości (którego prawdopodobieństwo wynosi wtedy $1 - p_1$) to prawdopodobieństwo takiego ciągu symboli o długości $n + 1$ określone jest wzorem:

$$\boldsymbol{p(n) = p_1^n \cdot (1 - p_1)} \tag{3.8}$$

Ostatnie wyrażenie jest matematycznym opisem funkcji **geometrycznego rozkładu prawdopodobieństwa**, i dla takiego właśnie rozkładu statystycznego danych kodowanie Golomba jest optymalne. Pierwotna motywacja opracowania kodów Golomba – czyli efektywne kodowanie powtórzeń tego samego symbolu danych – może zostać tutaj z powodzeniem rozszerzona na bardziej ogólny przypadek efektywnego kodowania całkowitoliczbowych wartości $n \geq 0$, których prawdopodobieństwo wystąpienia $p(n)$ jest właśnie takie jak wyrażone wzorem 3.8.

Można powiedzieć, że wzór 3.8 definiuje całą rodzinę krzywych prawdopodobieństwa (a nie tylko jeden wykres prawdopodobieństwa) przypominających rozkład wykładniczy. To, z którym konkretnie wykresem funkcji prawdopodobieństwa $p(n)$ mamy do czynienia (spośród całej rodziny wykresów) zależy od wartości parametru p_1, co zobrazowano na kilku wykresach rysunku 3-10.



Rysunek 3-10. Funkcje opisujące geometryczny rozkład prawdopodobieństwa danych. Krzywe rozkładu prawdopodobieństwa dla czterech wybranych wartości parametru p_1.

$$(p_1 = 0{,}25;\ p_1 = 0{,}5;\ p_1 = 0{,}75;\ p_1 = 0{,}85)$$

Aby kodowanie było efektywne dla każdego z przedstawionych powyżej rozkładów statystycznych danych (czyli dla dowolnego p_1), należy odpowiednio dobrać w projektowanym kodzie postać oraz długości słów kodowych jakie przypisuje się poszczególnym symbolom danych. W przypadku kodowania Golomba robi się to dobierając we właściwy sposób wartość parametru m kodu Golomba, nazywanego jego rzędem. Rząd m kodu Golomba wpływa istotnie na cechy tworzonego kodu: postać słów kodowych oraz ich długość, co zostanie w dalszej części tekstu zobrazowane odpowiednimi przykładami.

W kodach Golomba wyróżnić można dwie części: **przedrostek kodu** oraz jego **przyrostek**. Dla danej całkowitej wartości $n \geq 0$ oraz określonego rzędu m kod Golomba można utworzyć realizując kroki następującego algorytmu [Sayo00, Salom10]:

- Utwórz kod unarny dla wartości $q = \left\lfloor \frac{n}{m} \right\rfloor$. Utworzony kod unarny jest przedrostkiem tworzonego kodu Golomba.
- Wyznacz wartości: $r = n - qm$ oraz $c = \lceil log_2 m \rceil$. W oparciu o wyznaczone wartości tworzony jest przyrostek kodu Golomba:
  - Jeśli $r < 2^c - m$ przyrostek kodu stanowi binarna reprezentacja liczby r kodowanej na $c - 1$ bitach.
  - Jeśli $r \geq 2^c - m$ przyrostek kodu stanowi binarna reprezentacja liczby $(r + 2^c - m)$ kodowanej na c bitach.
- Wynikowy kod Golomba rzędu m jest połączeniem wyznaczonych wcześniej dwóch kodów: przedrostkowego oraz przyrostkowego.

(**Uwaga:** W podanym algorytmie operacja $\lfloor x \rfloor$ wyznacza część całkowitą liczby x, natomiast operacja $\lceil x \rceil$ oznacza najmniejszą liczbę całkowitą, która jest większa lub równa x)

Tabela 3.1 przedstawia kody Golomba rzędu $m = 1,2,3,\ldots,12$ dla wybranych wartości liczby n.

Tabela 3.1 Kody Golomba dla $m = 1,2,3,\ldots,12$ dla wybranych wartości liczby n. Symbolem | oddzielono część przedrostkową kodu od części przyrostkowej. Kody wyznaczone z użyciem własnego oprogramowania.

c	0	1	2	2	3	3	3	3	4	4	4	4
2^c - m	0	0	1	0	3	2	1	0	7	6	5	4

		Kody Golomba										
kodowana wartość **n**	**m=1**	**m=2**	**m=3**	**m=4**	**m=5**	**m=6**	**m=7**	**m=8**	**m=9**	**m=10**	**m=11**	**m=12**
0	0|	0|0	0|0	0|00	0|00	0|00	0|00	0|000	0|000	0|000	0|000	0|000
1	10|	0|1	0|10	0|01	0|01	0|01	0|010	0|001	0|001	0|001	0|001	0|001
2	110|	10|0	0|11	0|10	0|10	0|100	0|011	0|010	0|010	0|010	0|010	0|010
3	1110|	10|1	10|0	0|11	0|110	0|101	0|100	0|011	0|011	0|011	0|011	0|011
4	11110|	110|0	10|10	10|00	0|111	0|110	0|101	0|100	0|100	0|100	0|100	0|1000
5	111110|	110|1	10|11	10|01	10|00	0|111	0|110	0|101	0|101	0|101	0|1010	0|1001
6	1111110|	1110|0	110|0	10|10	10|01	10|00	0|111	0|110	0|110	0|1100	0|1011	0|1010
7	11111110|	1110|1	110|10	10|11	10|10	10|01	10|00	0|111	0|1110	0|1101	0|1100	0|1011
8	111111110|	11110|0	110|11	110|00	10|110	10|100	10|010	10|000	0|1111	0|1110	0|1101	0|1100
9	1111111110|	11110|1	1110|0	110|01	10|111	10|101	10|011	10|001	10|000	0|1111	0|1110	0|1101
10	11111111110|	111110|0	1110|10	110|10	110|00	10|110	10|100	10|010	10|001	10|000	0|1111	0|1110
11	111111111110|	111110|1	1110|11	110|11	110|01	10|111	10|101	10|011	10|010	10|001	10|000	0|1111
12	1111111111110|	1111110|0	11110|0	1110|00	110|10	110|00	10|110	10|100	10|011	10|010	10|001	10|000
13	11111111111110|	1111110|1	11110|10	1110|01	110|110	110|01	10|111	10|101	10|100	10|011	10|010	10|001
14	111111111111110|	11111110|0	11110|11	1110|10	110|111	110|100	110|00	10|110	10|101	10|100	10|011	10|010
15	1111111111111110|	11111110|1	111110|0	1110|11	1110|00	110|101	110|010	10|111	10|110	10|101	10|100	10|011
16	11111111111111110|	111111110|0	111110|10	11110|00	1110|01	110|110	110|011	110|000	10|1110	10|1100	10|1010	10|1000

Strona celowo pozostawiona pusta.

Analiza zawartości tej tabeli pokazuje jak wartość parametru m w kodzie Golomba wpływa na postać oraz długość kolejnych słów kodowych. I tak, w przypadku małej wartości parametru m (np. $m = 2$) pierwsze kody Golomba (zdefiniowane dla małych wartości n) są krótkie, i ich długość szybko rośnie dla coraz większych wartości kodowanej liczby n. Sytuacja odwrotna ma miejsce, kiedy parametr m przyjmuje dużą wartość (patrz kody w tabeli kodów dla $m = 12$). W tej sytuacji pierwsze kody są stosunkowo długie, ale wraz z dalszym wzrostem wartości kodowanej liczby n następuje już bardzo powolny przyrost długości kodu. W związku z tymi obserwacjami, wyznaczone przy małej wartości parametru m kody Golomba należy stosować w sytuacjach, kiedy prawdopodobieństwo małej/małych wartości n jest stosunkowo wysokie przy jednoczesnym małym prawdopodobieństwie występowania wartości dużych (omawiany przypadek odzwierciedla na rysunku 3-10 wykres funkcji prawdopodobieństwa dla $p_1 = 0{,}25$). W przypadkach bardziej wyrównanych wartości prawdopodobieństw kolejnych liczb n (patrz wykres funkcji prawdopodobieństwa dla $p_1 = 0{,}85$ na rysunku 3-10) bardziej opłacalne jest stosowanie kodów Golomba wysokiego rzędu (duża wartość parametru m).

Przytoczone rozumowanie nie pozwala jednak jeszcze na dobór właściwego rzędu m kodu Golomba, w obliczu znanego rozkładu statystycznego kodowanych danych (znana wartość p_1 w wyrażeniu 3.8). Szczegółowe badania Gallagera oraz van Voorhisa [Gall75] doprowadziły do wyznaczenia tej zależności – optymalna wartość m dla kodu Golomba wynosi:

$$\boldsymbol{m = \left\lceil -\frac{log_2(1+p_1)}{log_2(p_1)} \right\rceil} \tag{3.9}$$

Zgodnie z tym wzorem, dla wartości liczbowych, których częstość występowania odpowiada rozkładowi prawdopodobieństwa dla $p_1 = 0{,}25$ należy zastosować kody Golomba rzędu m=1. W przypadku takiego jak dla $p_1 = 0{,}85$ rozkładu statystycznego danych optymalne będą kody 4-tego rzędu (m=4).

Złożoność obliczeniowa kodowania Golomba jak również **dekodowania** kodu wynikają wprost z kroków przedstawionego wcześniej algorytmu wyznaczania słów kodowych Golomba. Analizując poszczególne kroki algorytmu widać, że realizuje się operacje: dzielenia (tylko w koderze), mnożenia, odejmowania, logarytmowania oraz przesunięcia bitowego wartości 1 o zadaną liczbę bitów (ostatnia operacja odpowiada podniesieniu liczby 2 do potęgi będącej wartością całkowitą dodatnią – patrz algorytm tworzenia kodu Golomba). Niektóre z wymienionych operacji można stablicować (np. operację logarytmu), co może być źródłem przyspieszenia obliczeń.

Wyznaczenie w **koderze** kodu Golomba dla całkowitej wartości $n \geq 0$ sprowadza się do utworzenia dwóch kodów dla obliczonych wcześniej wartości ilorazu (wartość q) oraz wartości reszty (wartość r). Dla wartości q wyznacza się kod unarny, co obliczeniowo jest bardzo proste. Otrzymujemy w ten sposób przedrostek kodu Golomba. Bardziej złożone jest wyznaczenie przyrostka kodu (kod dla wartości r), bo w zależności od wartości r kod ten może mieć różną długość (patrz szczegóły algorytmu tworzenia kodu Golomba). Celem ustalenia długości przyrostka kodu Golomba, wykonać należy operację porównania odpowiednich liczb (patrz algorytm).

Celem **zdekodowania** wartości $n \geq 0$ dekoduje się najpierw przedrostek kodu, który jest kodem unarnym. To dekodowanie jest zatem bardzo proste, a jego wynikiem jest wartość ilorazu q. Następnie realizuje się bardziej obliczeniowo złożone dekodowanie przyrostka kodu, prowadzące do odtworzenia wartości reszty r. Wyższa złożoność dekodowania kodu przyrostkowego wynika

z faktu, że kod ten może przyjmować długość $c-1$ bitów lub c bitów. Dlatego najpierw czyta się ze strumienia $c-1$ bitów, otrzymując w ten sposób wartość reszty r. Jeśli $r < 2^c - m$, wtedy dekodowanie przyrostka kodu jest już zakończone. Dekodowana wartość n może być wyznaczona ze wzoru: $n = r + qm$. W przeciwnym razie (czyli $r \geq 2^c - m$), dekodowanie wartości r nie zostało jeszcze zakończone. Należy w tym przypadku pobrać dodatkowo jeszcze jeden bit ze strumienia, który ‘doklejony’ do pobranych wcześniej $c-1$ bitów kończy dekodowanie wartości r. W tym przypadku dekodowaną wartość n wylicza się ze wzoru: $n = r - (2^c - m) + qm$.

3.2.5.3. Kodowanie Rice’a (Golomba-Rice’a)

Kodowanie Rice’a jest szczególnym przypadkiem kodowania Golomba – jest to przypadek, kiedy rząd m kodu Golomba jest potęgą liczby 2 (czyli $m = 2^0, 2^1, 2^2, 2^3, \ldots$). Z tego powodu kody Rice’a są czasami w literaturze określane mianem kodów Golomba-Rice’a. Kody Rice’a dziedziczą większość cech kodów Golomba, o których była mowa w poprzednim punkcie. Taka sama pozostaje również generalna idea tworzenia kodu dla kodowanej wartości $n \geq 0$, jak również dekodowania kodu. Kody Rice’a rzędu $m = 1,2,4,8$ dla wybranych wartości n kodowanej liczby znaleźć można w tabeli 3.1.

Z uwagi na ograniczenie na dozwoloną wartość parametru m w kodzie Rice’a, upraszcza się w **koderze** procedura tworzenia przyrostka kodu (w stosunku do przedstawionego wcześniej sposobu tworzenia przyrostka kodu dla kodu Golomba). W tym przypadku przyrostek kodu ma ściśle określoną długość i wynosi ona c bitów, a nie jak w przypadku kodu Golomba jedną z dwóch długości: $c-1$ lub c bitów. Ułatwia to również **dekodowanie** kodów Rice’a. Po odtworzeniu wartości ilorazu q (dekodowanie unarne) oraz wyliczeniu wartości reszty r (pobranie c bitów ze strumienia bitowego) dekodowana wartość n wyliczana jest zgodnie z zależnością: $n = r + qm$.

Obecnie, kody Rice’a znajdują praktyczne zastosowanie w bezstratnej kompresji dźwięku [FLAC].

3.2.5.4. Kodowanie Exp-Golomba *k*-tego rzędu

Kody Exp-Golomba (czyli wykładnicze kody Golomba) opracowane zostały w roku 1978 przez Teuholę [Teuh78] i stanowią modyfikację przedstawionych wcześniej kodów Golomba. W stosunku do oryginalnego kodu Golomba, kod Exp-Golomba opisany jest innymi wzorami na wartości q oraz r, przez co zmianie uległ rozkład prawdopodobieństwa danych, dla którego kod ten jest optymalny. Analogicznie jak wcześniej, kody Exp-Golomba składają się z dwóch części: przedrostka kodu oraz przyrostka kodu. Dla danej całkowitej wartości $n \geq 0$ kod Exp-Golomba m-tego rzędu można utworzyć realizując następujący algorytm [Teuh78, Marp03a]:

- Utwórz kod unarny dla wartości $q = \left\lfloor log_2\left(\frac{n}{2^m} + 1\right)\right\rfloor$. Utworzony kod unarny jest przedrostkiem tworzonego kodu Exp-Golomba.
- Podaj binarną reprezentację liczby $r = n + 2^m(1 - 2^q)$ wykorzystując $m + q$ znaczących bitów. Utworzony w ten sposób kod jest przyrostkiem tworzonego kodu Exp-Golomba.
- Wynikowy kod Exp-Golomba m-tego rzędu jest połączeniem wyznaczonych wcześniej kodów: przedrostkowego oraz przyrostkowego.

(**Uwaga:** Operacja $\lfloor \ldots \rfloor$ w powyższym algorytmie oznacza zaokrąglenie wyniku w dół do liczby całkowitej.)

Realizując wyżej przedstawiony algorytm dla wartości $n = 5$ oraz przy założeniu 0-ego rzędu kodu Exp-Golomba ($m = 0$) otrzymamy kod 11010 (przedrostek kodu: 110, przyrostek kodu: 10), natomiast kod 1110010 będzie w takim przypadku odpowiadał wartości $n = 9$. W analogiczny sposób wyznaczyć można kody Exp-Golomba dla innych wartości n oraz przy założeniu innych niż $m = 0$ rzędów kodu. Dla większej jasności tematu w tabeli 3.2 przedstawiono kody Exp-Golomba rzędu $m = 0,1,2,3$ dla wybranych wartości n.

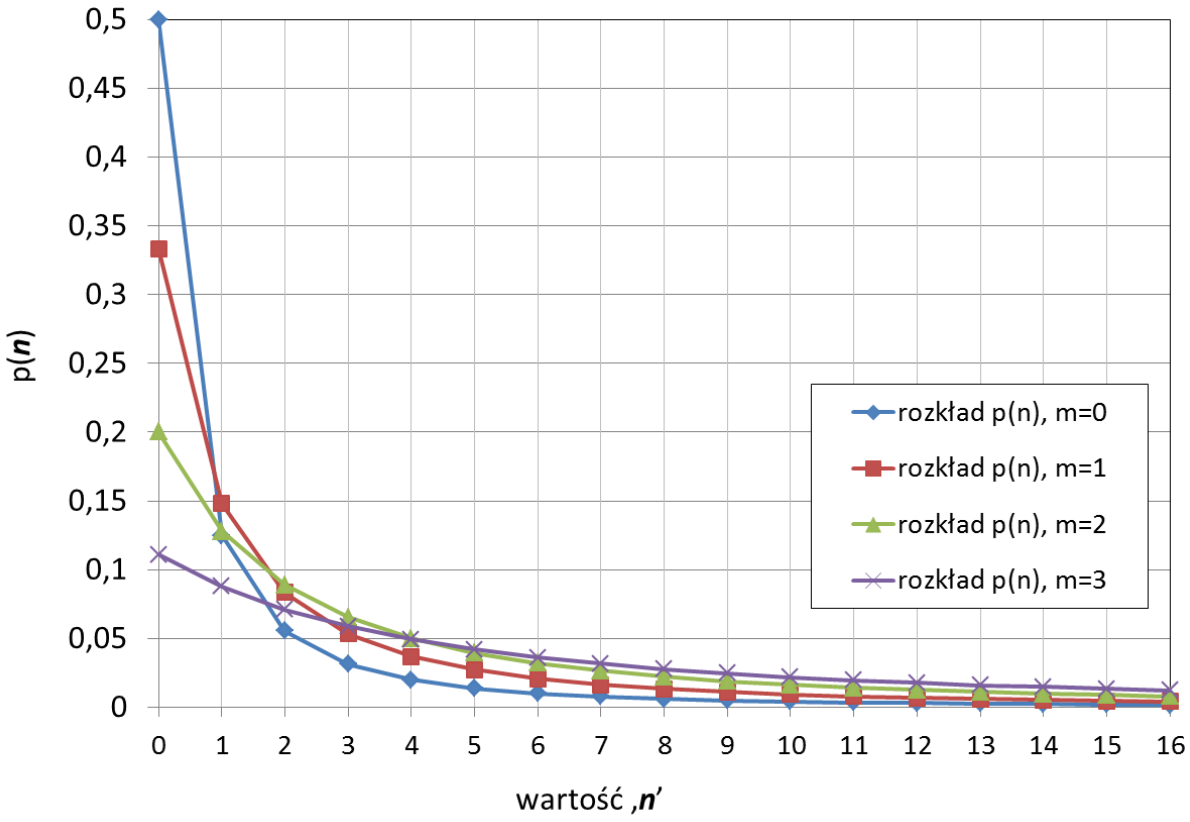
Tabela 3.2 Kody Exp-Golomba rzędu $m = 0,1,2,3$ dla wybranych wartości liczby n. Symbolem | oddzielono część przedrostkową kodu od części przyrostkowej. Kody wyznaczone z użyciem własnego oprogramowania.

	Kody Exp-Golomba			
kodowana wartość **n**	**m=0**	**m=1**	**m=2**	**m=3**
0	0|	0|0	0|00	0|000
1	10|0	0|1	0|01	0|001
2	10|1	10|00	0|10	0|010
3	110|00	10|01	0|11	0|011
4	110|01	10|10	10|000	0|100
5	110|10	10|11	10|001	0|101
6	110|11	110|000	10|010	0|110
7	1110|000	110|001	10|011	0|111
8	1110|001	110|010	10|100	10|0000
9	1110|010	110|011	10|101	10|0001
10	1110|011	110|100	10|110	10|0010
11	1110|100	110|101	10|111	10|0011
12	1110|101	110|110	110|0000	10|0100
13	1110|110	110|111	110|0001	10|0101
14	1110|111	1110|0000	110|0010	10|0110
15	11110|0000	1110|0001	110|0011	10|0111
16	11110|0001	1110|0010	110|0100	10|1000
17	11110|0010	1110|0011	110|0101	10|1001
18	11110|0011	1110|0100	110|0110	10|1010
19	11110|0100	1110|0101	110|0111	10|1011
20	11110|0101	1110|0110	110|1000	10|1100
21	11110|0110	1110|0111	110|1001	10|1101
22	11110|0111	1110|1000	110|1010	10|1110
23	11110|1000	1110|1001	110|1011	10|1111
24	11110|1001	1110|1010	110|1100	110|00000
25	11110|1010	1110|1011	110|1101	110|00001
26	11110|1011	1110|1100	110|1110	110|00010

27	11110|1100	1110|1101	110|1111	110|00011
28	11110|1101	1110|1110	1110|00000	110|00100
29	11110|1110	1110|1111	1110|00001	110|00101
30	11110|1111	11110|00000	1110|00010	110|00110
31	111110|00000	11110|00001	1110|00011	110|00111
32	111110|00001	11110|00010	1110|00100	110|01000

Jak wynika z algorytmu tworzenia kodu Exp-Golomba, jego długość wynosi $l = m + 2 \cdot q + 1$. Zgodnie z teorią Shannona, minimalna (czyli optymalna) długość kodu (wyrażana w bitach) jaka jest niezbędna do bezstratnego zakodowania symbolu danych występującego z prawdopodobieństwem p wynosi $log_2\left(\frac{1}{p}\right)$. Zestawiając tę optymalną długość kodu z faktyczną długością l kodu Exp-Golomba m-tego rzędu wyprowadzić można postać rozkładu statystycznego danych, dla którego kodowanie Exp-Golomba jest optymalne (tzn., że nie da się uzyskać wyższej efektywności kodowania). Rozkład ten wyraża się poniższym wzorem (wzór wyznaczony przez autora) i dla wybranych wartości parametru m został zobrazowany na rysunku 3-11:

$$\boldsymbol{p(n) = \frac{1}{2^{m+1} \cdot \left(\frac{n}{2^m}+1\right)^2}} \tag{3.10}$$

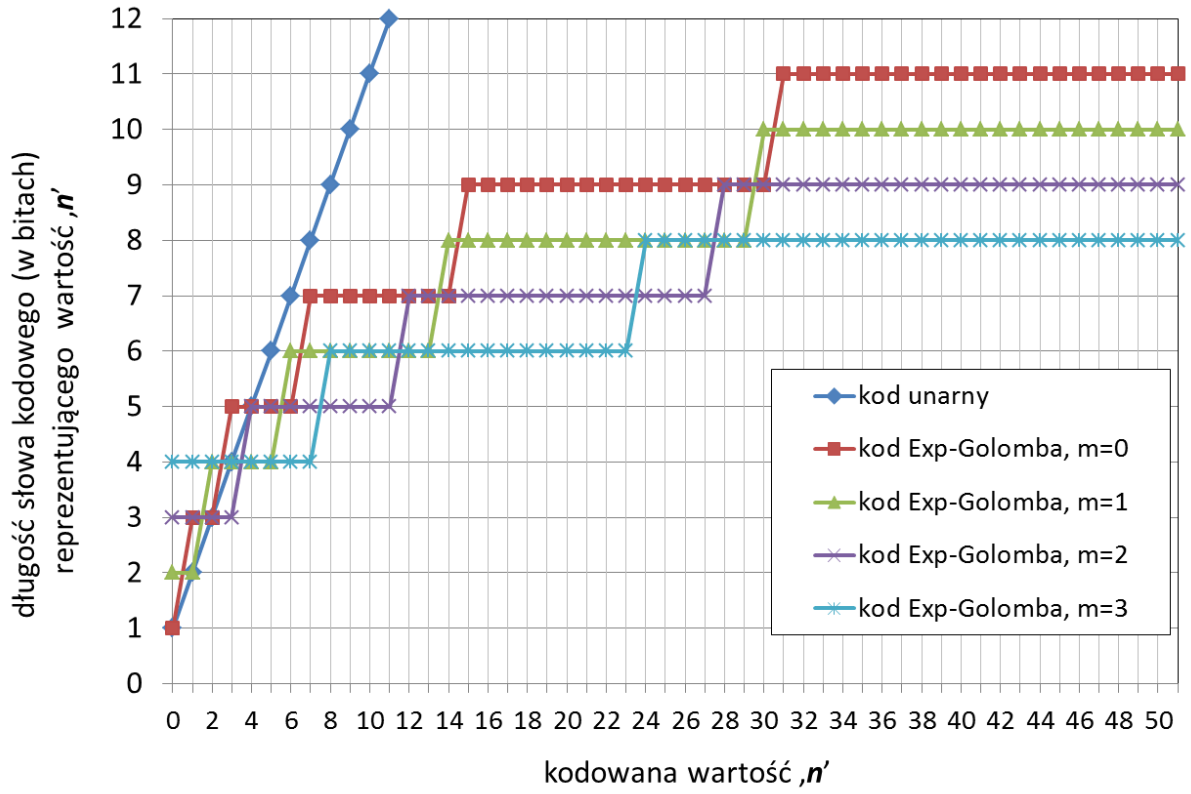


Rysunek 3-11. Funkcje wykładniczego rozkładu prawdopodobieństwa. Krzywe rozkładu dla czterech wybranych wartości parametru m ($m = 0, m = 1, m = 2, m = 3$).

Tak więc, kodowanie Exp-Golomba jest efektywne dla danych o wykładniczym rozkładzie prawdopodobieństwa. Rozkład taki jest charakterystyczny dla danych reprezentujących sygnał

błędu predykcji (czyli dane resztkowe), który jest generowany przez dobrze działający predyktor danych. Stąd praktyczne zastosowanie tych kodów we współczesnych standardach kompresji stratnej obrazu, np. MPEG-4 AVC/H.264 [AVC] oraz AVS [AVS], celem efektywnej reprezentacji wybranych danych resztkowych w koderze.

Kody Exp-Golomba mają istotną własność polegającą na tym, że wraz ze wzrostem długości l kodu liczba różnych kodów o tej długości rośnie wykładniczo. To znaczy, że wraz ze wzrostem długości l kodu wykładniczo rośnie liczba wartości (liczb), które kodem o tej długości można reprezentować. Wynika z tego bardzo pożyteczna cecha kodu, zgodnie z którą, wraz ze wzrostem wartości kodowanej liczby n następuje tylko logarytmiczny (a nie liniowy jak ma to miejsce w przypadku kodu unarnego) wzrost długości kodu, który pozwala tę liczbę reprezentować. Przedstawione na rysunku 3-12 porównanie długości kodu unarnego oraz wybranych kodów Exp-Golomba dobrze obrazuje różnicę w długości tych kodów dla kolejnych wartości n.



Rysunek 3-12. Długości wybranych kodów uniwersalnych.

Jak widać, kodowanie unarne nie jest efektywne w przypadku danych, których rozkład prawdopodobieństwa jest wykładniczy. W tym konkretnym przypadku, liniowe wydłużanie się kodu unarnego dla rosnących wartości n powoduje, że kodowanie dużych wartości obarczone jest bardzo dużym kosztem bitowym (kod unarny o dużej długości).

3.3. Kodowanie arytmetyczne

3.3.1. Wprowadzenie

Omawiając techniki kodowania danych, w których wykorzystuje się kodowanie o zmiennej długości słowa kodowego wykazano, że możliwe jest zwiększenie efektywności kompresji w przypadku kodowania bloków (ciągów) symboli danych, zamiast poszczególnych symboli niezależnie (patrz **blokowe kodowanie Huffmana**). Zwiększoną efektywność kodowania danych uzyskuje się jednak za cenę zwiększonej złożoności obliczeniowej oraz złożoności pamięciowej algorytmu kodowania i dekodowania. Problem zwiększonej złożoności algorytmu jest tym większy im dłuższe bloki (ciągi) symboli danych się tworzy, dla których realizuje się następnie kodowanie i dekodowanie danych.

Istnieje jednak inna technika kodowania entropijnego danych, której teoretyczna efektywność jest bardzo zbliżona do efektywności techniki blokowego kodowania Huffmana. Mowa tutaj o **kodowaniu arytmetycznym** danych.

Idea kodowania arytmetycznego danych (ale tylko bardzo ogólnie rozumiana idea) jest bardzo podobna do idei blokowego kodowania Huffmana, w którym słowo kodowe przypisuje się do bloku (ciągu) symboli, a nie do poszczególnych symboli danych niezależnie [Witt87, Riss79, Doma98, Sayo00, Przel05, Salom06]. W tym sensie, kodowanie arytmetyczne może być również rozumiane jako kodowanie o zmiennej długości słowa kodowego. Należy jednak podkreślić, że mechanizm tworzenia słowa kodowego jest tutaj mocno inny od sposobu tworzenia kodu w blokowym kodowaniu Huffmana.

Na czym zatem polega różnica? W przypadku **blokowego kodowania Huffmana** kolejne bloki (ciągi) symboli danych koduje się z użyciem wyznaczonych wcześniej słów kodowych (ciągów bitów) – czyli słowa kodowe przypisywane są poszczególnym blokom symboli. Liczba symboli w jednym bloku (ciągu) ustalana jest przed rozpoczęciem kodowania danych, i w praktyce nie może być ona duża (głównie z uwagi na pamięciową złożoność algorytmu). **Kodowanie arytmetyczne** natomiast, sprowadza się do przypisania słowa kodowego (pewnego ciągu bitów) dla całej sekwencji kodowanych symboli danych (czyli np. dla całej sekwencji symboli zawartej w kodowanym pliku), a nie na przypisywaniu słów kodowych dla kolejnych bloków (ciągów) symboli o niewielkiej długości. Dodatkowo, w kodowaniu arytmetycznym słowo kodowe, które będzie reprezentować całą sekwencję kodowanych symboli wyznaczane jest w koderze sukcesywnie, i co najważniejsze nie są, jak ma to miejsce w blokowym kodowaniu Huffmana, tworzone słowa kodowe dla alternatywnych sekwencji symboli danych. Ostatni z wymienionych elementów jest istotną zaletą kodowania arytmetycznego, gdyż pozwala uniknąć poważnego problemu, jakim w blokowym kodowaniu Huffmana jest konieczność przechowywania w pamięci kodera i dekodera tablic słów kodowych o dużych rozmiarach. Przytoczone tutaj porównanie idei kodowania arytmetycznego z blokowym kodowanie Huffmana jest pewnym uproszczeniem, jednak zdaniem autora pozwala lepiej zrozumieć zasadniczą różnicę pomiędzy omawianymi algorytmami.

Zakodowanie (ale również zdekodowanie) sekwencji symboli danych z użyciem techniki kodowania arytmetycznego wymaga wykonania dużej ilości obliczeń. Złożoność obliczeniowa kodowania arytmetycznego jest znacząco wyższa niż złożoność kodowania Huffmana. Jeszcze do niedawna był to jeden z głównych powodów (poza ograniczeniami wynikającymi z praw patentowych do metody) dość rzadkiego praktycznego zastosowania technik kodowania arytmetycznego dla celów kompresji danych. Jednak później, opracowanie szybkich implementacji kodowania arytmetycznego [Penn88, Taub02, Marp03b] jak również ogromny wzrost dostępnej w procesorach mocy obliczeniowej sprawiły, że kodowanie arytmetyczne stało się bardzo atrakcyjne.

W systemach kompresji danych (również danych multimedialnych), jakie opracowano w ostatnich latach obserwuje się wzrost zainteresowania technikami kodowania arytmetycznego. Kodowanie arytmetyczne jest już częścią systemów archiwizacji danych [Mah05], kompresji obrazów statycznych (standard JPEG 2000 [JPEG2000]), ale również współczesnych technik kodowania ruchomego obrazu (standardy H.263, AVC/H.264, HEVC [H263, AVC, Marpe03a, Richa03, HEVC]).

3.3.2. Kodowanie arytmetyczne – algorytm podstawowy

Unikalne słowo kodowe reprezentujące całą sekwencję kodowanych symboli danych jest wyznaczane w koderze z uwzględnieniem prawdopodobieństwa pojawienia się w strumieniu danych poszczególnych symboli. Tak więc, przed przystąpieniem do kodowania arytmetycznego danych wyznaczyć trzeba statystykę danych, które mają zostać zakodowane, jeśli prawdopodobieństwa wystąpienia kolejnych symboli nie są jeszcze znane. Załóżmy zbiór N symboli alfabetu $S = \{x_1, x_2, \dots, x_N\}$, których prawdopodobieństwa wystąpienia wynoszą odpowiednio $P = \{p(x_1), p(x_2), \dots, p(x_N)\}$. Mówiąc najogólniej, w koderze arytmetycznym, każdy z symboli alfabetu, jak również sekwencja zakodowanych do tej pory symboli danych reprezentowane są pewnymi przedziałami liczbowymi o ściśle określonych granicach (początek oraz koniec przedziału liczbowego). Długości przedziałów liczbowych wynikają wprost z prawdopodobieństw występowania symboli/sekwencji symboli, które są w koderze reprezentowane przez te przedziały liczbowe.

Aby zrealizować powyższą ideę należy wykonać odpowiednie obliczenia w koderze arytmetycznym, które można opisać następującymi krokami postępowania [Witt87, Riss79, Sayo00, Doma98, Sayo00, Przel05, Salom06]:

Krok 1. W koderze definiowany jest przedział liczbowy $\langle 0, 1)$.

Krok 2. Przedział liczbowy $\langle 0, 1)$ dzielony jest na N mniejszych fragmentów/części. Długości poszczególnych fragmentów są równe prawdopodobieństwom $p(x_k)$ $(k = 1,2, \dots, N)$ występowania poszczególnych symboli danych. Poszczególnym symbolom x_k $(k = 1,2, \dots, N)$ przyporządkowuje się odpowiednie fragmenty przedziału $\langle 0, 1)$ w taki sposób, aby długości fragmentów przyporządkowanych do symboli były równe prawdopodobieństwom $p(x_k)$ $(k = 1,2, \dots, N)$ występowania tych symboli. Wraz z kodowaniem pierwszego symbolu x_m, wybierany jest skojarzony z tym symbolem właściwy fragment przedziału $\langle 0, 1)$. Wybrany fragment staje się aktualnym przedziałem liczbowym.

Krok 3. Koder wczytuje kolejny symbol x_n. Aktualny przedział liczbowy ponownie dzielony jest na fragmenty/części. Od tej pory jednak, długości poszczególnych fragmentów są proporcjonalne[27] (a nie równe jak w Kroku 2) do prawdopodobieństw $p(x_k)$ $(k = 1,2, \dots, N)$ występowania symboli danych. Ponownie, poszczególnym symbolom x_k $(k = 1,2, \dots, N)$ przyporządkowuje się odpowiednie fragmenty aktualnego przedziału liczbowego (podobnie jak miało to miejsce w Kroku 2). Fragment aktualnego przedziału skojarzony z aktualnie kodowanym symbolem x_n staje się nowym aktualnym przedziałem.

[27] Tak więc, jeśli prawdopodobieństwo wystąpienia kodowanego symbolu wynosi 0,3 (czyli 30% - gdyby wyrazić częstość wystąpienia w procentach) to z kodowanym symbolem będzie skojarzony pewien fragment przedziału, którego długość będzie stanowić 30% długości aktualnego przedziału liczbowego.

Krok 4. Krok 3 obliczeń jest powtarzany do momentu zakodowania wszystkich danych. Za każdym razem wyznaczane są nowe granice aktualnego przedziału liczbowego.

Krok 5. Uzyskany w drodze kodowania ostatniego symbolu aktualny przedział liczbowy jest wynikiem kodowania arytmetycznego. Binarna reprezentacja dowolnej liczby należącej do tego przedziału jest słowem kodowym (ciągiem bitów), które w sposób jednoznaczny reprezentuje całą sekwencję symboli wejściowych. Chociaż z punktu widzenia efektywności kodowania powinno wybierać się liczbę o najkrótszym słowie binarnym, to w praktyce liczbą tą jest najczęściej lewy koniec wynikowego przedziału liczbowego lub jego środek.

Ponieważ w przedstawionym tutaj algorytmie kodowaniu podlegały symbole danych pochodzące z *N*-elementowego alfabetu $S = \{x_1, x_2, \dots, x_N\}$, kodowanie takie jest nazywane *N*-arnym kodowaniem arytmetycznym, a rdzeń kodera arytmetycznego, który realizuje ten algorytm rdzeniem *N*-arnym.

3.3.3. Kodowanie arytmetyczne – algorytm podstawowy w ujęciu matematycznym

W poprzednim punkcie skupiono się na metodzie kodowania arytmetycznego widzianej niejako z „lotu ptaka”. W tym punkcie algorytm kodowania arytmetycznego zostanie jeszcze raz powtórzony, ale z uwzględnieniem operacji matematycznych, jakie wykonuje koder kodując dane.

Krok 1. Definiowany jest przedział liczbowy $\langle 0, 1)$.

Krok 2. Przedział liczbowy $\langle 0, 1)$ dzielony jest na *N* mniejszych fragmentów/części. Otrzymane fragmenty przedziału $\langle 0, 1)$ przypisuje się symbolom wejściowego alfabetu, wykonując działania matematyczne przedstawione w tabeli 3.3.

Tabela 3.3 Przyporządkowanie przedziałów liczbowych poszczególnym symbolom.

Symbol	Prawdopodobieństwo symbolu	Przyporządkowany symbolowi przedział liczbowy
x_1	p_1	$\langle 0, p_1)$
x_2	p_2	$\langle p_1, p_2)$
…	…	…
x_N	p_N	$\left\langle \sum_{k=1}^{N-1} p_k , 1\right)$

Jeśli pierwszym kodowanym symbolem jest symbol x_m, wybierany jest m-ty fragment przedziału $\langle 0, 1)$, który został przyporządkowany temu symbolowi. W ten sposób przedział $\langle a, b)$ staje się aktualnym przedziałem liczbowym, gdzie $a = \sum_{k=1}^{m-1} p_k$ oraz $b = \sum_{k=1}^{m} p_k$.

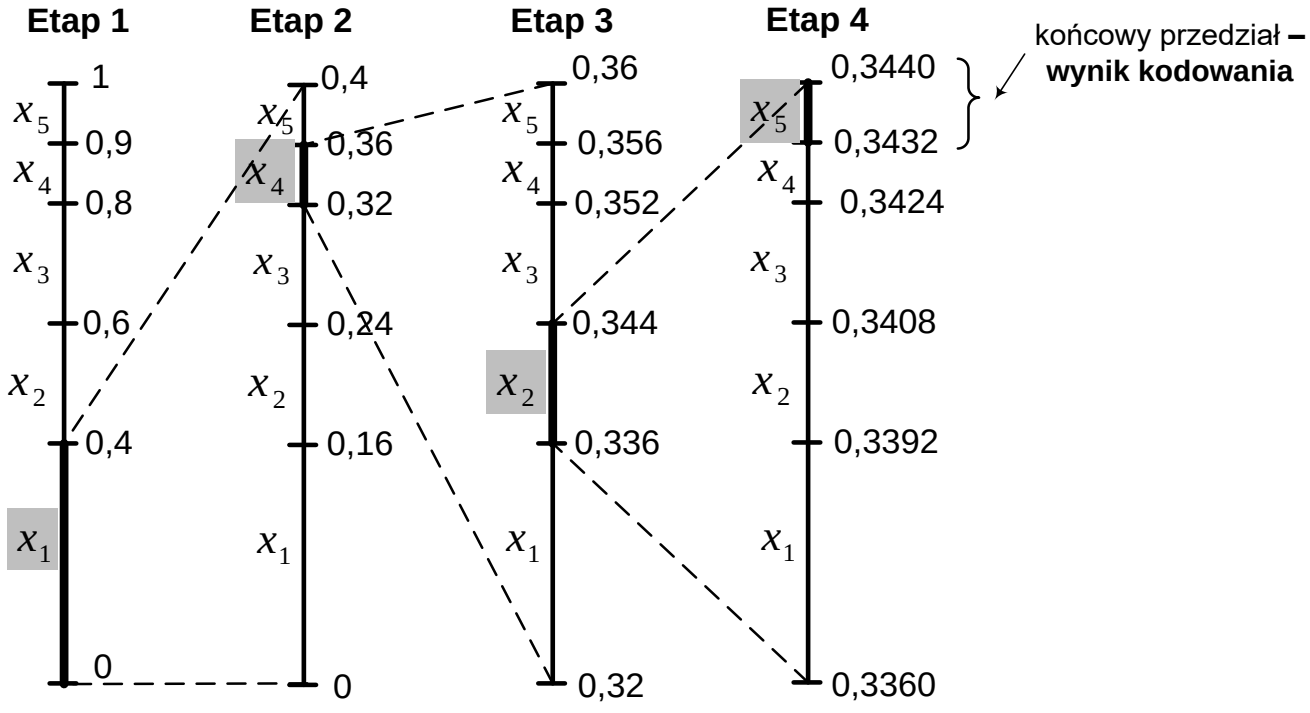
Krok 3. Koder wczytuje kolejny symbol x_n. Aktualny przedział liczbowy $\langle a, b)$ ponownie dzielony jest na fragmenty/części – długości poszczególnych fragmentów wynikają z prawdopodobieństw występowania symboli danych. Fragmentem przedziału $\langle a, b)$, który będzie skojarzony z aktualnie

kodowanym symbolem x_n jest fragment $\langle a + (b-a)\cdot c, a + (b-a)\cdot d)$, gdzie c oraz d określają granice przedziału liczbowego, który został przypisany symbolowi x_n w kroku 2 algorytmu (czyli $c = \sum_{k=1}^{n-1} p_k$ oraz $d = \sum_{k=1}^{n} p_k$). Nowy przedział $\langle a, b)$ jest ustalany realizując następujące podstawienia: $a \leftarrow a + (b-a)\cdot c$ oraz $b \leftarrow a + (b-a)\cdot d$. Otrzymany w ten sposób przedział $\langle a, b)$ staje się aktualnym przedziałem liczbowym.

Krok 4. Krok 3 obliczeń jest powtarzany do momentu zakodowania wszystkich symboli danych. Za każdym razem wyznaczane są nowe granice $\langle a, b)$ aktualnego przedziału liczbowego.

Krok 5. Uzyskany po zakodowaniu ostatniego symbolu przedział $\langle a, b)$ jest wynikiem kodowania arytmetycznego sekwencji wejściowych symboli danych. Binarna reprezentacja dowolnej liczby należącej do tego przedziału jest słowem kodowym (ciągiem bitów) reprezentującym w sposób jednoznaczny całą sekwencję symboli wejściowych.

Celem lepszego zrozumienia przedstawionego algorytmu na rysunku 3-13 przedstawiono rezultaty kolejnych etapów obliczeń w przypadku kodowania przykładowej, krótkiej sekwencji wejściowych symboli: $x_1x_4x_2x_5$. Przyjęto tutaj założenie, że symbole pochodzą z pięcio-elementowego alfabetu $S = \{x_1, x_2, x_3, x_4, x_5\}$ i pojawiają się w strumieniu danych z prawdopodobieństwami odpowiednio $P = \{0{,}4; 0{,}2; 0{,}2; 0{,}1; 0{,}1\}$.



Rysunek 3-13. Główna idea kodowania arytmetycznego. Wynik kodowania sekwencji pięciu symboli: $x_1x_4x_2x_5$.

Binarna reprezentacja dowolnej liczby należącej do przedziału $\langle 0{,}3432;\ 0{,}3440)$ jest słowem kodowym reprezentującym sekwencję $x_1x_4x_2x_5$. Liczba **0,34375** należy do tego przedziału, a jej binarna reprezentacja to **0,01011**. Otrzymana liczba będzie zawsze mniejsza od wartości 1, bo należy

ona do przedziału, który zawiera się w początkowym przedziale $\langle 0;\ 1)$. Z tego powodu zerowy bit znajdujący się przed przecinkiem liczby **0,01011** nie musi być uwzględniany, przesyłany do dekodera. Ostatecznie więc, do dekodera wysyłane jest słowo kodowe **01011** i reprezentuje ono w tym przypadku sekwencję symboli $x_1x_4x_2x_5$.

3.3.4. Dekodowanie arytmetyczne danych – krótka dyskusja

Otrzymana w wyniku kodowania arytmetycznego **liczba** należąca do wynikowego przedziału liczbowego umożliwia jednoznaczne zdekodowanie sekwencji zakodowanych symboli danych. Jest tak dlatego, gdyż na każdym etapie obliczeń koder przyporządkowuje poszczególnym symbolom danych rozłączne przedziały liczbowe, a wynikowy przedział liczbowy całkowicie zawiera się w początkowym przedziale $\langle 0;\ 1)$.

Dlatego też, żeby zdekodować kolejne symbole zakodowanej sekwencji dekoder musi otrzymać **liczbę**, która jest wynikiem kodowania oraz dodatkowo, powtórzyć obliczenia, jakie w poszczególnych etapach wykonał koder. Ideę działania dekodera można łatwo zrozumieć analizując kolejne etapy obliczeń dekodera celem zdekodowania rozważanej w poprzednim punkcie sekwencji: $x_1x_4x_2x_5$ (patrz rysunek 3-13). Dla większej jasności rozważania przeprowadzone zostaną w notacji dziesiętnej liczb.

Dekoder rozpoczyna swoją pracę od powtórzenia obliczeń kodera wykonanych w ramach **etapu 1** (patrz **etapy** zaznaczone na rysunku 3-13). W ten sposób dekoder uzyska wiedzę o sposobie przyporządkowania symboli alfabetu do przedziałów liczbowych. Mając wynik tych obliczeń dekoder sprawdza, w którym z otrzymanych przedziałów liczbowych zawiera się liczba **0,34375** (będąca wynikiem kodowania). W tym przypadku liczba **0,34375** jest częścią przedziału $\langle 0;\ 0{,}4)$, który został skojarzony z symbolem x_1 – symbol x_1 jest zatem wynikiem dekodowania przeprowadzonego w pierwszym etapie obliczeń. Dekoder przystępuje do **etapu 2** obliczeń. Podział przedziału liczbowego $\langle 0;\ 0{,}4)$ na fragmenty daje wynik jak na rysunku 3-13 (etap 2). Liczba **0,34375** zawiera się w przedziale $\langle 0{,}32;\ 0{,}36)$, który został przypisany symbolowi x_4 – symbol ten jest wynikiem dekodowania przeprowadzonego w etapie 2. Do tej pory dekoderowi udało się zdekodować sekwencję symboli: x_1x_4. W **etapie 3** obliczeń aktualny przedział liczbowy $\langle 0{,}32;\ 0{,}36)$ (skojarzony wcześniej z symbolem x_4) jest dzielony na fragmenty/części, czego wynikiem są przedstawione na rysunku przedziały liczbowe (patrz 3-13, etap 3). Liczba **0,34375** jest częścią skojarzonego z symbolem x_2 przedziału $\langle 0{,}336;\ 0{,}344)$, dlatego symbol x_2 jest wynikiem dekodowania w etapie 3. W **etapie 4** obliczeń aktualny przedział liczbowy $\langle 0{,}336;\ 0{,}344)$ jest dalej dzielony na fragmenty, czego wynikiem jest kilka mniejszych przedziałów liczbowych (patrz 3-13, etap 4). Liczba **0,34375** należy do przedziału $\langle 0{,}3432;\ 0{,}3440)$, który reprezentuje symbol x_5 – jest on zatem wynikiem dekodowania przeprowadzonego w tym etapie obliczeń. W ten sposób dekoder pomyślnie zdekodował całą sekwencję symboli: $x_1x_4x_2x_5$.

3.3.5. Efektywność techniki kodowania arytmetycznego

W przypadku kodowania arytmetycznego sekwencji n symboli danych średnia długość $\bar{L}$ słowa kodowego, która przypada na pojedynczy symbol danych wejściowych zawiera się w następujących granicach [Sayo00]:

$$H\{x_1, x_2, \dots, x_N\} \leq \bar{L} \leq H\{x_1, x_2, \dots, x_N\} + \frac{2}{n} \tag{3.11}$$

Porównując tę zależność z innym wzorem (wzór **3.4**), który opisuje efektywność podstawowej wersji kodowania Huffmana widać, że maksymalna średnia długość słowa kodowego, przypadająca na jeden symbol danych jest mniejsza w przypadku kodowania arytmetycznego, w przypadku, kiedy liczba n symboli w kodowanej sekwencji jest większa niż 2. W przedstawionym scenariuszu kodowanie arytmetyczne cechuje się zatem wyższą efektywnością kodowania, w porównaniu z podstawowym algorytmem kodowania Huffmana.

Istnieje jednak możliwość poprawienia efektywności kodowania Huffmana, realizując łączne kodowanie bloków (ciągów) symboli, zamiast kolejnych symboli danych niezależnie (patrz szczegóły na temat blokowego kodowania Huffmana). Blokowy koder Huffmana operujący na n elementowych blokach (ciągach) symboli cechuje się minimalnie wyższą efektywnością kodowania w porównaniu z kodowaniem arytmetycznym[28], o czym świadczą wzory 3.6 oraz 3.11 określające granice efektywności rozważanych technik kodowania danych. Tak jest w teorii. Z punktu widzenia zastosowań praktycznych trudno sobie jednak wyobrazić użycie blokowego kodowania Huffmana operującego na bardzo długich blokach (ciągach) symboli (czyli duża wartość n). Nie jest to możliwe w obliczu dużej złożoności obliczeniowej oraz ogromnej złożoności pamięciowej algorytmu Huffmana operującego na długich blokach danych. Stąd obserwowany w ostatnich latach wzrost zainteresowania technikami kodowania arytmetycznego danych.

3.3.6. Algorytm podstawowy kodowania arytmetycznego – dwa istotne problemy

Kolejne etapy obliczeń, jakie przeprowadzane są w koderze i dekoderze arytmetycznym generują coraz węższe przedziały liczbowe. Po zakodowaniu (ale również zdekodowaniu) zaledwie kilku symboli danych różnica dwóch liczb, opisujących odpowiednio dolną oraz górną granicę aktualnego przedziału liczbowego, może być bardzo mała. Dlatego konieczna jest tutaj bardzo duża precyzja wykonywanych obliczeń i bardzo dokładna reprezentacja liczb określających granice przedziałów liczbowych. W ogólnym przypadku kodowania dowolnie długiej sekwencji symboli wejściowych **precyzja ta musi być nieskończona.** W teorii nie jest to problemem, bo liczby możemy zapisywać z wykorzystaniem dowolnie dużej dokładności. Ale tylko w teorii. W kontekście aplikacji kodowania arytmetycznego i praktycznej realizacji kodeka (koder oraz dekoder), na przykład na procesorze, dysponujemy skończoną dokładnością reprezentacji liczb, co wynika z oczywistych ograniczeń architektury procesorów (ograniczona długość rejestrów procesora). Nie jest zatem możliwa realizacja podstawowego algorytmu na stałoprzecinkowym procesorze. Dostępna w wielu procesorach bardzo dokładna (ale też obliczeniowo bardzo kosztowna) zmiennoprzecinkowa reprezentacja liczb okaże się również niewystarczająca w przypadku kodowania odpowiednio długiej sekwencji symboli danych. Przedstawiony podstawowy algorytm kodowania arytmetycznego **nie daje się zatem zrealizować w praktyce**. Jest to **pierwszy istotny problem** omówionej metody.

W związku z powyższym przez wiele lat koder i dekoder arytmetyczny działały tylko „na papierze” i funkcjonowały tylko w sferze rozważań teoretycznych. Nie było algorytmów numerycznych opisujących działanie kodera i dekodera, które dałoby się zastosować w maszynach liczących. Tym samym, nie było implementacji programowych kodeka.

Drugim problemem, jaki wiąże się z algorytmem podstawowym jest opóźnienie kodowania i dekodowania sekwencji symboli. Otóż w koderze arytmetycznym wynikowe słowo

[28] wraz ze wzrostem *n* różnica efektywności jest jednak coraz mniejsza

kodowe generowane jest dopiero po przetworzeniu wszystkich symboli sekwencji wejściowej. Nie ma tutaj możliwości, żeby słowo kodowe wytwarzane było przez koder systematycznie, fragmentami, wraz z zakończeniem kodowania pewnej części wejściowej sekwencji symboli. Algorytm nie działa zatem w sposób przyrostowy, co jest przyczyną dużego opóźnienia kodowania danych. W przypadku dekodowania symboli wczytać należy całe słowo kodowe, które może być bardzo długie, bo długość ta wynika wprost z rozmiaru strumienia zakodowanych danych. Nie ma możliwości systematycznego wczytywania kolejnych fragmentów słowa kodowego i dekodowania fragmentów sekwencji symboli. Rodzi to duże opóźnienie przetwarzania danych i wiąże się z olbrzymią złożonością pamięciową algorytmu.

Z punktu widzenia praktycznej realizacji kodowania arytmetycznego istotne są zatem modyfikacje podstawowego algorytmu żeby: 1) możliwe stało się uruchomienie kodera i dekodera na stałoprzecinkowym procesorze, oraz 2) metody kodowania i dekodowania danych mogły działać w sposób przyrostowy.

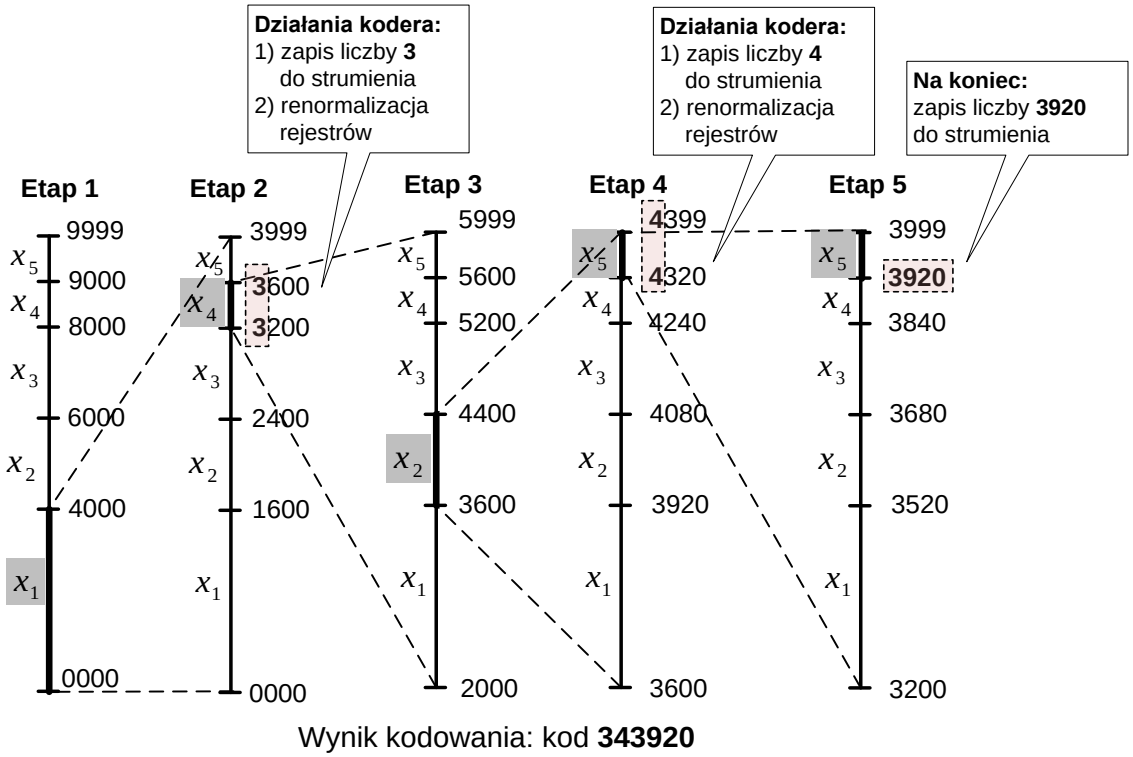
Rozwiązania spełniające wymienione wymagania zostały w 1976 roku przedstawione w niezależnych pracach Pasco i Rissanena [Pas76, Riss76]. Tak więc, miało to miejsce wiele lat po opracowaniu podstawowego algorytmu, i dopiero te rozwiązania otworzyły furtkę do praktycznego wykorzystania technik kodowania arytmetycznego dla celów kompresji danych.

3.3.7. „Prawie” praktyczna realizacja kodowania arytmetycznego

Z punktu widzenia praktycznej realizacji kodowania arytmetycznego istotne jest przejście z bardzo kosztownej ułamkowej (zmiennoprzecinkowej) reprezentacji liczb w kodeku na obliczeniowo znacznie prostszą reprezentację całkowitoliczbową (czyli stałoprzecinkową). Aby koder i dekoder arytmetyczny mogły działać w oparciu o obliczenia przeprowadzane na liczbach całkowitych (a nie ułamkach jak ma to miejsce w podstawowym algorytmie) początkowy przedział liczbowy $\langle 0, 1)$ musi zostać odpowiednio przeskalowany na przedział zawierający liczby całkowite. Przeskalowanie przedziału $\langle 0, 1)$ uzyskuje się mnożąc graniczne wartości przedziału (tutaj odpowiednio **0** oraz **1**) przez pewną stałą wartość całkowitoliczbową. Przemnażając na przykład liczby przedziału $\langle 0, 1)$ przez stałą wartość 10000 otrzymamy przedział $\langle 0, 10000)$. Uwzględniając fakt, że z prawej strony przedział ten jest otwarty, oraz dodatkowo chęć przeprowadzania obliczeń na liczbach całkowitych, prawy koniec przedziału $\langle 0, 10000)$ musi zostać zmniejszony o 1, i wtedy do dyspozycji mamy obustronnie domknięty przedział liczb całkowitych $\langle 0, 9999\rangle$. Aby móc w procesorze zapisać dowolną liczbę z tego przedziału potrzebna jest w tym przypadku 4-cyfrowa precyzja rejestrów procesora (liczona w notacji dziesiętnej liczb). Tak więc, stopień przeskalowania oryginalnego przedziału $\langle 0, 1)$ musi uwzględniać dostępną w procesorze precyzję rejestrów.

W każdym z etapów obliczeniowych koder oraz dekoder pamiętają tylko końce bieżącego przedziału liczbowego (lewy i prawy koniec), co wymaga w procesorze wykorzystania dwóch rejestrów: rejestru **L** przechowującego lewy koniec przedziału, oraz rejestru **H** przechowującego prawy koniec przedziału. W przypadku rozważanej tutaj 4-cyfrowej precyzji obliczeń początkowe wartości rejestrów będą więc następujące: **L**=0000, oraz **H**=9999. Początkowy przedział liczbowy $\langle 0000, 9999\rangle$ będzie dalej podlegał podziałowi na coraz mniejsze fragmenty/części wraz z kodowaniem/dekodowaniem kolejnych symboli danych. Wszystkie operacje matematyczne będą przeprowadzane w koderze i dekoderze z wykorzystaniem arytmetyki całkowitoliczbowej.

Algorytm kodowania operujący na całkowitych liczbach nie rozwiązuje jednak problemu precyzji obliczeń w koderze i dekoderze, a jeszcze ten problem pogłębia. Przedział $\langle L, H\rangle$ zawiera skończoną liczbę całkowitych wartości, przez co możliwe jest reprezentowanie relatywnie krótkiej sekwencji wejściowych symboli. Tak więc, algorytm ten ciągle nie może być obiektem praktycznej realizacji. Rozwiązaniem tego problemu jest przeprowadzana systematycznie w trakcie kodowania czy dekodowania danych **renormalizacja** (przeskalowanie) aktualnych wartości zapisanych w rejestrach **L** oraz **H**. Sama idea systematycznego przeskalowywania **L** i **H** jest bardzo prosta i została zobrazowana na rysunku 3-14. Zakodowanie (lub zdekodowanie) nowego symbolu powoduje zawężenie aktualnego przedziału liczbowego (opisanego wartościami **L** i **H**). Realizując kodowanie kolejnych symboli wartości **L** oraz **H** będą się ze sobą schodzić. Jeśli najbardziej znaczące cyfry w **L** i **H** będą takie same to znaczy, że wartość ta nie zmieni się już do końca procesu kodowania sekwencji (patrz etap 2 oraz etap 4 obliczeń na rysunku 3-14). Dlatego można tą wartość od razu zapisać do strumienia bitowego, po czym dokonać przeskalowania rejestrów **L** i **H** przesuwając ich zawartości o jedną cyfrę w lewo. Po przesunięciu zawartości rejestrów, najmłodszą pozycję **L** i **H** należy uzupełnić odpowiednio cyfrą **0** oraz **9**[29]. Otrzymaliśmy w ten sposób algorytm kodowania arytmetycznego, który bazuje na obliczeniach stałoprzecinkowych oraz na bieżąco zapisuje kolejne cyfry słowa kodowego do wyjściowego strumienia zakodowanych danych. Czyli algorytm działa w sposób przyrostowy. Dodatkowo, aktualny przedział liczbowy jest na bieżąco przeskalowywany w koderze i dekoderze, co eliminuje problem nieskończonej precyzji obliczeń, jaki występował w przypadku podstawowego algorytmu. Przedstawiony tutaj algorytm kodowania arytmetycznego jest już o krok od metody, która może być obiektem programowej realizacji.

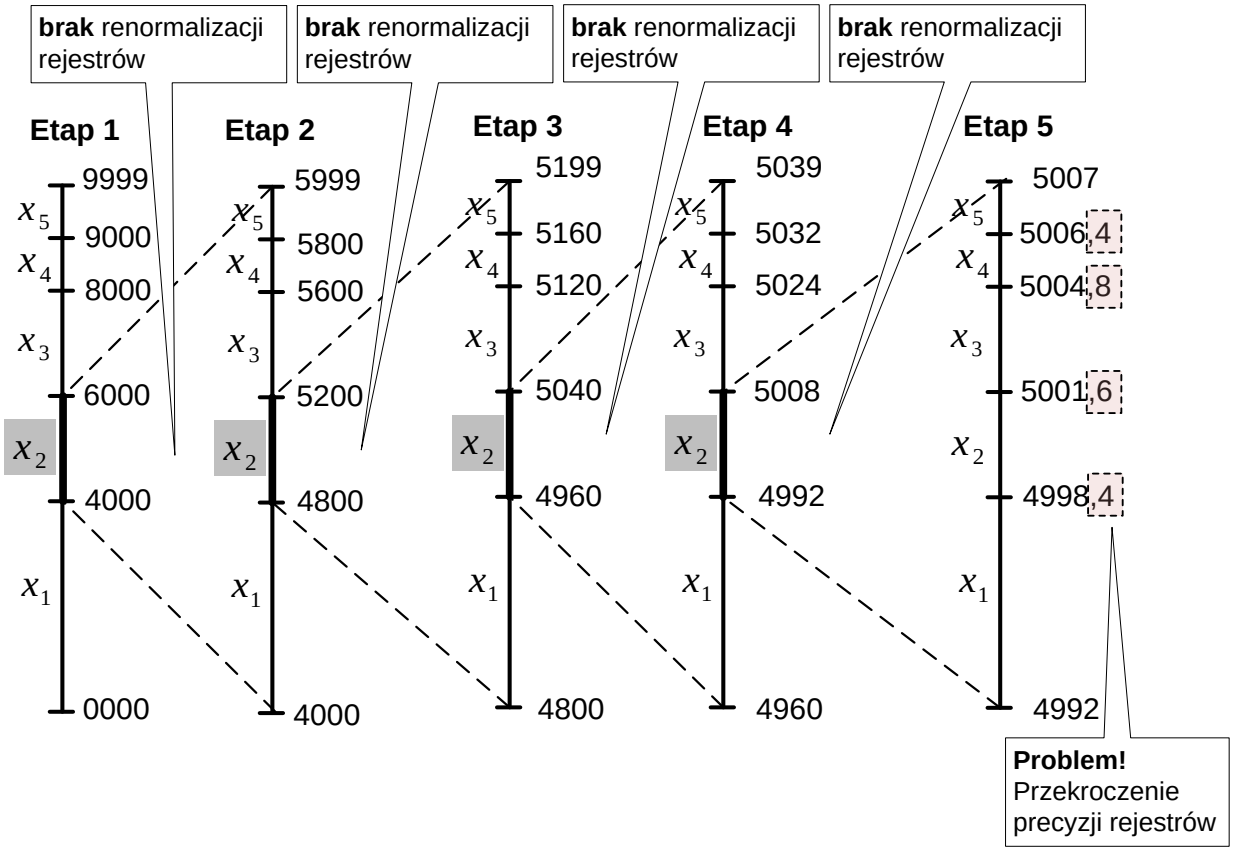


Rysunek 3-14. Idea kodowania arytmetycznego działającego w sposób przyrostowy i realizującego obliczenia w dziedzinie liczb całkowitych. Wynik kodowania sekwencji symboli $x_1x_4x_2x_5x_5$.

[29] **Uwaga:** W etapie 2 obliczeń przedział, który reprezentuje symbol x_4 to $\langle 3200, 3599\rangle$. Widoczna na powyższym rysunku wartość 3600 jest już początkiem przedziału skojarzonego z symbolem x_5. Dlatego po wysunięciu do strumienia bitowego wartości 3 i po przesunięciu pozostałej zawartości rejestrów otrzymaliśmy nowy przedział o końcach $\langle 2000, 5999\rangle$.

3.3.8. Problem przekroczenia precyzji rejestrów w kodeku

Kontrola tylko i wyłącznie wartości najbardziej znaczącej cyfry występującej w rejestrach **L** i **H** kodeka nie zapewnia prawidłowego działanie algorytmu przedstawionego w poprzednim punkcie. Przy odpowiedniej kombinacji symboli wejściowych może się zdarzyć, że wraz z kodowaniem kolejnych symboli danych zapisane w **L** i **H** wartości będą się do siebie „zbliżać”, a jednocześnie najstarsze cyfry w **L** i **H** będą inne. W takim przypadku nie zostanie wywołana procedura przeskalowania (renormalizacji) rejestrów oraz procedura zapisu do strumienia najbardziej znaczącej cyfry. Z uwagi na ustaloną z góry skończoną precyzję rejestrów **L** oraz **H** brak renormalizacji rejestrów doprowadzi w kolejnych etapach obliczeń do przekroczenia precyzji reprezentacji liczb, przez co najmłodsza część aktualnego słowa kodowego zacznie być w koderze tracona. Do zakończenia procedury kodowania kolejnych symboli koder nie zapisze już żadnej informacji do wyjściowego strumienia bitowego. W efekcie, dekoder nie będzie w stanie zrealizować dekodowania zakodowanych przez koder symboli. Opisywana sytuacja znana jest w literaturze pod pojęciem **problemu niedomiaru** (ang. underflow). Przykład takiej sytuacji został przedstawiony na rysunku 3-15.



Rysunek 3-15. Idea kodowania arytmetycznego działającego w sposób przyrostowy i realizującego obliczenia w dziedzinie liczb całkowitych. Ilustracja problemu niedomiaru.

Aby rozwiązać ten problem należy odpowiednio wcześniej przewidzieć taką sytuację i zawczasu przeprowadzić odpowiednie przeskalowanie rejestrów. Należy to zrobić w sytuacji, kiedy najbardziej znaczące cyfry rejestrów **L** i **H** nie są równe ale:

- różnią się o 1, i jednocześnie

- druga najbardziej znacząca cyfra rejestru **H** równa jest **0** a druga najbardziej znacząca cyfra rejestru **L** jest równa **9**.

(omawiany przypadek odpowiada wartościom **L**=**49**xx, **H**=**50**xx, jakie otrzymano w etapie 3 obliczeń w przedstawionym na rysunku 3-15 przykładzie)

W tym przypadku zawartości rejestrów **L** oraz **H** muszą zostać przesunięte o jeden w lewo, ale z pominięciem przesuwania najbardziej znaczącej cyfry, jaka znajduje się w rejestrach **L** i **H**. W ten sposób:

1. Najbardziej znacząca cyfra rejestrów **L** i **H** się nie zmienia, natomiast druga najbardziej znacząca cyfra rejestrów **L** oraz **H** jest nadpisywana cyfrą sąsiednią, która znajduje się na młodszej pozycji, oraz dodatkowo:
2. Zwiększany jest licznik **counter** wykonania tej operacji (uwaga: koder rozpoczyna pracę z zerowym stanem licznika **counter**).
3. Na najmłodszej pozycji rejestru **L** wpisywana jest cyfra **0**, a najmłodsza pozycja rejestru **H** przyjmuje wartość **9**.
4. Jeśli po wykonaniu powyższych operacji nadal występuje problem niedomiaru, całą procedurę jego usunięcia należy powtórzyć.

Wynikiem zastosowania procedury usunięcia niedomiaru w omawianym przypadku będzie stan rejestrów: **L**=4600 i **H**=5409. Po usunięciu niedomiaru koder kontynuuje kodowanie kolejnych symboli sekwencji wejściowej.

Jak dotąd, wystąpienie niedomiaru nie zostało jeszcze w żaden sposób zasygnalizowane w strumieniu wyjściowym. Zostanie to zrobione przy okazji najbliższego wywołania procedury zapisu części słowa kodowego do strumienia bitowego – czyli w momencie kiedy najbardziej znaczące cyfry rejestrów **L** i **H** staną się równe. W omawianym przykładzie najbardziej znacząca cyfra w **L** oraz **H** przyjmie wartość **4** lub **5** (co zależeć będzie od sekwencji kodowanych symboli). W tej sytuacji do strumienia wyjściowego zapisana zostanie wartość tej najbardziej znaczącej cyfry (czyli tutaj **4** lub **5**), i dodatkowo pewna liczba cyfr, które informować będą o pojawieniu się problemu niedomiaru. Jakie będą wartości tych cyfr i ile ich będzie? Liczba cyfr sygnalizujących niedomiar wynika wprost z wartości licznika **counter** (czyli z krotności wystąpienia problemu niedomiaru od ostatniego zerowania licznika **counter**). Wartości tych cyfr zależą od relacji wartości rejestrów **L** i **H** mierzone w dwóch różnych momentach: 1) w momencie pierwszego wystąpienia niedomiaru od ostatniego wyzerowania licznika **counter** (czyli w omawianym przykładzie są to wartości **L**=**49**xx, **H**=**50**xx), oraz 2) aktualnych wartości **L** i **H**. W ten sposób widać czy w drodze kolejnych modyfikacji rejestrów (tutaj **L**=**49**xx, **H**=**50**xx) ich wartości podążały do liczby **4**xxx, czy do liczby **5**xxx. Zgodnie z tym tokiem rozumowania do strumienia zapisywane są następujące cyfry reprezentujące niedomiar:

- **counter** cyfr o wartości **9**, jeśli najstarszą cyfrą w rejestrach **L** i **H** jest aktualnie cyfra **4** lub
- **counter** cyfr o wartości **0**, jeśli najstarszą cyfrą w rejestrach **L** i **H** jest aktualnie cyfra **5**.

Po tych czynnościach wywołana zostanie procedura przeskalowania (renormalizacji) rejestrów **L** i **H**.

3.3.9. „Całkowitoliczbowa” realizacja kodowania arytmetycznego – końcowy algorytm

Algorytm kodowania arytmetycznego działający w oparciu o arytmetykę całkowitoliczbową jest następujący (tutaj algorytm zakładający 4-cyfrową precyzję obliczeń):

Krok 1. Ustaw początkowe wartości: **L**=0000, **H**=9999, oraz **counter**=0.

Krok 2. Wczytaj nowy symbol x_n. Dokonaj podziału aktualnego zakresu $\langle \mathbf{L}, \mathbf{H} \rangle$ na N fragmentów/części zgodnie z ideą kodowania arytmetycznego. Zaktualizuj wartości zakresu $\langle \mathbf{L}, \mathbf{H} \rangle$ w następujący sposób:

$$\begin{aligned} &przedział = L - H + 1 \\ &c = \sum_{k=1}^{n-1} p_k, \;\; d = \sum_{k=1}^{n} p_k \\ &L \leftarrow L + (przedział \cdot c) \\ &H \leftarrow L + (przedział \cdot d) \end{aligned} \tag{3.12}$$

Krok 3. Sprawdź wartości rejestrów **L** oraz **H** według następującej procedury:

Warunek 1: jeśli najbardziej znacząca cyfra w **L** i **H** są równe to wtedy:

- zapisz do strumienia wyjściowego wartość najbardziej znaczącej cyfry;
- przesuń zawartości obu rejestrów o jedną cyfrę w lewo;
- na najmłodszej pozycji **L** zapisz **0**, a na najmłodszej pozycji **H** zapisz **9**;
- jeśli **counter** > 0 wyślij odpowiednie dane sygnalizujące problem niedomiaru a następnie wyzeruj licznik **counter** (patrz problem niedomiaru w kodowaniu arytmetycznym);
- jeśli nadal zachodzi **warunek 1** to powtórz operacje powyżej.

Warunek 2: jeśli najbardziej znaczące cyfry **L** i **H** różnią się o 1 i jednocześnie druga najbardziej znacząca cyfra rejestru **H** równa jest **0**, a druga najbardziej znacząca cyfra rejestru **L** jest równa **9** to:

- przesuń o jeden w lewo zawartość rejestrów **L** i **H**, ale z pominięciem przesuwania najbardziej znaczącej cyfry w tych rejestrach (patrz rozwiązanie problemu niedomiaru);
- zwiększ licznik **counter** o 1;
- na najmłodszej pozycji **L** zapisz **0**, a na najmłodszej pozycji **H** zapisz **9**.
- jeśli nadal występuje niedomiar, powtórz czynności zdefiniowane w **warunku 2**.

Krok 4. Powtarzaj krok 2 aż do zakodowania wszystkich symboli wejściowej sekwencji.

Krok 4. Po zakodowaniu wszystkich symboli wyślij dodatkowo do strumienia stan rejestru **L**.

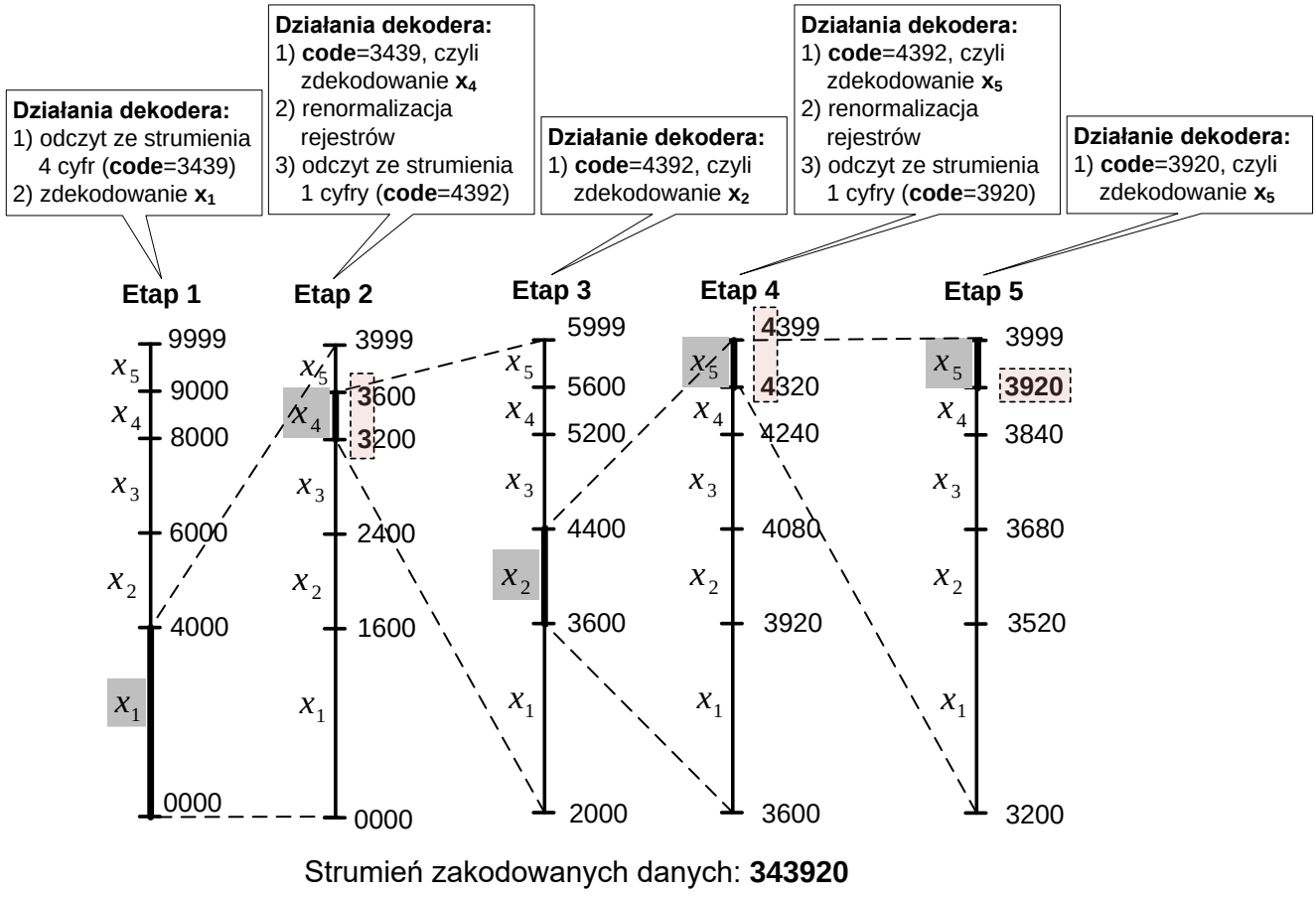
3.3.10. „Całkowitoliczbowa” realizacja dekodowania arytmetycznego

Wykonywane w dekoderze operacje są analogiczne do tych, jakie przeprowadza się po stronie kodera danych. Dlatego algorytm numeryczny dekodowania danych stanowi lustrzane odbicie algorytmu kodującego przedstawionego w poprzednim punkcie. Jedyna różnica pomiędzy

działaniem kodera i dekodera polega na tym, że zamiast systematycznego zapisywania do strumienia wyjściowego kolejnych cyfr słowa kodowego (co jest robione po stronie kodera) w dekoderze następuje systematyczne wczytywanie ze strumienia kolejnych cyfr słowa kodowego, celem zdekodowania symboli zakodowanej sekwencji. W związku z powyższym, tak samo jak ma to miejsce w koderze, dekoder operuje na rejestrach/zmiennych:

- **L** – dolna granica aktualnego przedziału liczbowego,
- **H** – górna granica aktualnego przedziału, oraz
- **counter** – licznik krotności wystąpienia problemu niedomiaru od momentu ostatniego wyzerowania tego licznika (patrz problem niedomiaru w kodowaniu arytmetycznym).

Dodatkowo, w dekoderze należy zdefiniować rejestr **code**, który w kolejnych etapach obliczeniowych przechowuje kolejne fragmenty słowa kodowego odczytywanego na bieżąco ze strumienia zakodowanych danych. Przed rozpoczęciem dekodowania sekwencji danych rejestr ten jest zapełniany pierwszymi cyframi słowa kodowego, jakie zostało wytworzone przez koder – liczba wczytywanych cyfr odpowiada przyjętej wcześniej w koderze i dekoderze precyzji przeprowadzania obliczeń (w przedstawionym na rysunku 3-16 przykładzie są to 4 cyfry zapisane w notacji dziesiętnej liczb). Kolejne kroki postępowania dekodera są już takie jak w koderze (ze wskazaną powyżej różnicą) i nie wymagają zdaniem autora osobnego komentarza. Działanie dekodera danych operującego w arytmetyce liczb całkowitych zostało dodatkowo zilustrowane przykładem (patrz rysunku 3-16).



Rysunek 3-16. „Całkowitoliczbowa” realizacja dekodowania arytmetycznego. Ilustracja dekodowania sekwencji symboli: $x_1 x_4 x_2 x_5 x_5$.

Chociaż dekoder arytmetyczny można zrealizować w opisany powyżej sposób, to w praktyce dokonywany na każdym etapie obliczeń dokładny podział aktualnego przedziału liczbowego na fragmenty nie jest tutaj konieczny (a wiadomo, że pochłania pewną liczbę cykli procesora). Zamiast tego wystarczy w odpowiedni sposób „przeskalować” zapisaną w rejestrze **code** wartość do nowej liczby **base_code** w taki sposób, żeby analizować położenie tej liczby w ramach początkowego przedziału $\langle 0000, 9999 \rangle$ (taki będzie początkowy przedział przy założeniu 4-cyfrowej precyzji obliczeń), a nie aktualnego przedziału określonego bieżącymi wartościami **L** i **H**. Takie „przeskalowanie” jest możliwe realizując następujący wzór ogólny:

$$base\_code = \left\lfloor \frac{code - L}{H - L + 1} \times 10^m \right\rfloor \tag{3.13}$$

gdzie m określa liczbę cyfr znaczących w rejestrach dekodera (czyli precyzję obliczeń – tutaj $m = 4$), natomiast operacja $\lfloor ... \rfloor$ realizuje zaokrąglenie wyniku w dół do liczby całkowitej.

W przedstawionym na rysunku 3-16 przykładzie, wyznaczone w kolejnych etapach wartości **code** zostaną przeliczone na wartości **base_code**:

- **base_code**=3439 (dla code=3439, L=0, H=9999), co w odniesieniu do początkowego przedziału liczbowego $\langle 0000, 9999 \rangle$ odpowiada symbolowi x_1 (patrz rysunek etapu 1),
- **base_code**=8597 (dla code=3439, L=0, H=3999), co w odniesieniu do początkowego przedziału $\langle 0000, 9999 \rangle$ odpowiada symbolowi x_4,
- **base_code**=5980 (dla code=4392, L=2000, H=5999), co w odniesieniu do początkowego przedziału $\langle 0000, 9999 \rangle$ odpowiada symbolowi x_2,
- **base_code**=9900 (dla code=4392, L=3600, H=4399), co w odniesieniu do początkowego przedziału $\langle 0000, 9999 \rangle$ odpowiada symbolowi x_5,
- **base_code**=9000 (dla code=3920, L=3200, H=3999), co w odniesieniu do początkowego przedziału $\langle 0000, 9999 \rangle$ odpowiada symbolowi x_5.

3.3.11. Algorytm kodowania arytmetycznego w zapisie dwójkowym

Celem lepszego zrozumienia omawianego tematu, algorytmy kodowania i dekodowania arytmetycznego danych zostały przedstawione z wykorzystaniem dziesiętnej reprezentacji liczb. Wykorzystywane w praktyce programowe oraz sprzętowe realizacje kodowania arytmetycznego danych operują jednak w systemie dwójkowym (binarnym). Bez względu jednak na przyjęty system liczbowy, w którym kodek realizuje obliczenia, schemat (idea) działania kodeka jest taki sam, z tą różnicą, że największą w systemie dziesiętnym cyfrę **9** należy zastąpić największą w systemie dwójkowym cyfrą **1**. Uwzględnienie tej jednej różnicy w przedstawionych wcześniej algorytmach kodowania i dekodowania arytmetycznego pozwoli uzyskać algorytm kodeka, który bazuje na dwójkowym systemie liczb.

3.3.12. Szybkie realizacje kodowania arytmetycznego

Z punktu widzenia zastosowań praktycznych istotną wadą omówionej „całkowitoliczbowej” realizacji kodowania arytmetycznego są operacje mnożenia i dzielenia liczb, które wykonuje się w każdym z kolejnych etapów obliczeniowych algorytmu. W stosunku do obliczeniowo relatywnie prostych operacji dodawania i odejmowania liczb, operacje mnożenia i dzielenia obarczone są większym kosztem (szczególnie w kontekście procesorów starszych generacji), i tym samym zwiększają one złożoność kodowania i dekodowania danych. Przez wiele

lat (w zasadzie aż do początku lat 90-tych), był to jeden z powodów niewielkiego praktycznego wykorzystania technik kodowania arytmetycznego w systemach kompresji danych.

W tym kontekście kluczowe stały się badania nad modyfikacją istniejących algorytmów numerycznych w taki sposób, żeby operacje mnożenia i dzielenia liczb zastąpić prostszymi od nich operacjami dodawania, odejmowania oraz przesunięcia bitowego wartości zapisanych w rejestrach. W tym celu opracowano warianty algorytmu, w których przedział liczbowy dzielono tylko na dwa fragmenty (co było obliczeniowo znacznie łatwiejsze), a nie jak wcześniej na większą liczbę części. Mowa tutaj o tzw. **binarnym kodowaniu arytmetycznym**, które operuje na danych pochodzących z dwuelementowego alfabetu (czyli kodujemy każdorazowo jeden z dwóch symboli: **0** lub **1**). Algorytm binarnego kodowania arytmetycznego stał się obiektem dalszych prac ukierunkowanych na dalszą optymalizacją działania kodeka.

Kamieniem milowym w rozwoju metod kodowania arytmetycznego danych było opracowanie szybkiej implementacji kodeka binarnego, o nazwie Q-kodek [Penn88]. Dzięki zastosowaniu w kodeku dodatkowych uproszczeń algorytm ten nie wymagał wykonywania kosztownych mnożeń i dzieleń, przez co otworzyła się droga do aplikacji kodeka w systemach kompresji danych. Opracowane później modyfikacje Q-kodeka, tzw. QM-kodek [Taub02] oraz MQ-kodek [Taub02] stały się częścią międzynarodowych standardów kompresji danych obrazowych: JBIG [JBIG], JPEG [JPEG], JBIG2 [JBIG2], JPEG2000 [JPEG2000, Taub02, Achar05].

Najbardziej obecnie (rok 2018) zaawansowanym kodekiem binarnym jest M-kodek [Marp03b, Marp06b], opracowany po roku 2000 jako element techniki kontekstowego adaptacyjnego binarnego kodowania arytmetycznego CABAC[30] (ang. Context-based Adaptive Binary Arithmetic Coding). W stosunku do wymienionych wcześniej szybkich realizacji kodeków binarnych, M-kodek cechuje się najniższą złożonością obliczeniową. Przedstawione w literaturze badania wydajności obliczeniowej kodeków wskazują na 5%-18% różnicę wydajności pomiędzy M-kodekiem a MQ-kodekiem na korzyść tego pierwszego [Marp04]. Co ciekawe, M-kodek jest również bardzo wydajny z punktu widzenia stopnia kompresji danych. Wytwarzany przez M-kodek zakodowany strumień danych jest zwykle mniejszy o 2%-4% w stosunku do rozmiaru strumienia obserwowanego na wyjściu MQ-kodeka [Marp04]. Eksperymentalnie wykazano, że w zastosowaniach do kompresji obrazu ruchomego, efektywność M-kodeka jest praktycznie taka sama jak efektywność tradycyjnego kodeka binarnego, który bazuje na czasochłonnych operacjach mnożenia i dzielenia liczb [Marp06a].

3.3.13. Modelowanie statystyczne danych w kodowaniu arytmetycznym – krótka dygresja

W przedstawionych do tej pory algorytmach pominięto bardzo ważną, z punktu widzenia efektywności kompresji, kwestię sposobu wyliczania w koderze i dekoderze prawdopodobieństw symboli danych. Założono, że wartości prawdopodobieństw wystąpienia symboli są już znane. W tym kontekście przedstawione wcześniej algorytmy prezentowały działanie samego **rdzenia kodera i dekodera arytmetycznego** danych, pomijając kwestię **statystycznego modelowania kodowanych danych.** Należy mocno podkreślić, że na cały koder/dekoder arytmetyczny składają się dwa elementy: 1) mechanizm estymacji prawdopodobieństw kodowanych symboli, oraz 2)

[30] Technika CABAC należy do najbardziej efektywnych metod kodowania entropijnego i jest stosowana w standardach MPEG-4 AVC/H.264 oraz MPEG-H HEVC/H.265 kompresji ruchomego obrazu.

rdzeń kodera/dekodera dokonujący kompresji/dekompresji tych danych (opisany dokładnie w poprzednich punktach książki).

Aby możliwa stała się efektywna reprezentacja wejściowych danych, koder entropijny musi dysponować wiedzą o strukturze tych danych, znać zależności, jakie występują między kolejnymi symbolami danych. Pozyskanie tej wiedzy jest właśnie zadaniem mechanizmu statystycznego modelowania danych. W najnowszych koderach arytmetycznych wiedza o strukturze kodowanych danych jest wyrażana wartościami **prawdopodobieństw warunkowych** wystąpienia poszczególnych symboli danych. Stopień zgodności wyznaczonych wartości prawdopodobieństw z faktyczną statystyką kodowanych danych determinuje efektywność arytmetycznego kodowania danych. W przypadku kodowania danych, których statystyka ulega ciągłym zmianom (a tak jest w przypadku danych reprezentujących obraz) trafne określenie prawdopodobieństw poszczególnych symboli jest bardzo trudne. W toku prac naukowych opracowanych zostało wiele metod efektywnej estymacji statystyki danych, np. metoda ważenia drzew kontekstów (ang. Context-Tree Weighting Method) [Will95, Will98a, Begl04], metoda predykcji poprzez częściowe dopasowanie (ang. Prediction with Partial Matching) [Clear84, Begl04], metody bazujące na modelach Markowa *k*-tego rzędu [Sayo00], czy metody wykorzystujące predefiniowane maszyny stanów (ang. Finite State Machine) [Marp03a, Marp04, Richa03]. Każda z opracowanych metod modelowania statystycznego zakłada konkretny model (charakter) danych – dokładność estymacji prawdopodobieństw jest tym większa, im bardziej założony model danych jest zgodny z faktycznym charakterem danych wejściowych.

W związku z powyższym stosowane w najnowszych koderach arytmetycznych (rok 2018) mechanizmy statystycznego modelowania danych są bardzo rozbudowane. Żeby dobrze zrozumieć wagę tej części kodera, bardzo wartościowa jest tutaj analiza (choćby uproszczona) metod, które stosuje się w koderze arytmetycznym CABAC [Marp03a, Marp04, Richa03]. Koder ten (wraz z opracowanymi po roku 2003 ulepszeniami [Mrak03a, Mrak03b, Mrak03c, Ghan04, Mila06, Bely06, Hong04, Firo06, Karw06, Karw07a, Karw07b, Karw08]) stanowi odzwierciedlenie aktualnego stanu techniki i nauki w zakresie arytmetycznego kodowania danych i został zaprojektowany z myślą o efektywnej reprezentacji danych obrazowych w koderach wizyjnych najnowszej generacji (MPEG-4 AVC/H.264 oraz MPEG-H HEVC/H.265) [AVC, HEVC]. W przytoczonych koderach wizyjnych kodowaniu entropijnemu podlegają dane: 1) sygnał błędu predykcji próbek obrazu, 2) informacja o ruchu w sekwencji (wektory ruchu, indeksy ramek odniesienia, itd.), oraz 3) dane sterujące (w tym sygnalizacja trybów kodowania użytych przez koder wizyjny w poszczególnych fragmentach obrazu). Chociaż dane te wykazują silną niestacjonarność (statystyki tych danych zmieniają się zarówno w ramach jednego obrazu (zmienność w przestrzeni), jak również w ramach kolejnych obrazów (zmienność w czasie)), to istnieje jednak pewna korelacja wartości sąsiadujących ze sobą symboli danych. Dodatkowo, statystyki symboli, które wchodzą w skład danych różnych typów (np. współczynniki transformaty błędu predykcji czy dane reprezentujące wektory ruchu) nie są takie same, więc statystyki tych danych należałoby wyznaczać niezależnie, a nie w sposób łączny. Dlatego, w celu zwiększenia efektywności kompresji, zastosowany w algorytmie CABAC mechanizm modelowania statystycznego symboli danych uwzględnia przytoczone cechy danych wejściowych (czyli danych podlegających kodowaniu w koderze entropijnym). W efekcie, cały strumień wejściowych danych dzielony jest na dużą liczbę mniejszych, niezależnych strumieni (w praktyce kilkaset takich strumieni) zgodnie ze ściśle zdefiniowanymi w algorytmie regułami. Mówiąc najprościej, to, do którego strumienia danych zostanie przypisany dany symbol, zależy od typu danych, z którym ten symbol jest związany oraz od kontekstu, czyli wartości kodowanego symbolu w sąsiednich blokach obrazu. Realizowane jest zatem **kontekstowe kodowanie danych**. W każdym z otrzymanych w ten sposób strumieni danych koder dokonuje niezależnej estymacji prawdopodobieństw warunkowych symboli wraz z uaktualnieniem wartości tych prawdopodobieństw co symbol. Oprócz kodowania kontekstowego następuje zatem **adaptacja**

działania kodera tak, żeby uwzględnić charakter lokalnych danych. Wszystko to sprawia, że mechanizm statystycznego modelowania danych, jaki zastosowano w algorytmie CABAC jest bardzo skomplikowany i znacząco wpływa na złożoność całego kodeka arytmetycznego. Sam rdzeń kodeka arytmetycznego stanowi współcześnie niewielką część całego arytmetycznego kodeka danych. W oprogramowaniu kodeka CABAC (liczącego **kilka tysięcy** linii kodu programu napisanego w języku C/C++), około **90%** wszystkich linii to fragment programu związany z estymacją prawdopodobieństw symboli danych [Karw08]. Sam rdzeń kodeka arytmetycznego to zaledwie **10%** całej implementacji kodeka. Przytoczone tutaj wartości liczbowe podkreślają ogromne znaczenie metod estymacji statystyki danych we współczesnych kodera arytmetycznych.

3.4. Efektywność kompresji oraz złożoność współczesnych technik kodowania entropijnego

Wymienione w tytule punktu parametry kodeka entropijnego zależą wprost od zastosowanych w nim algorytmów. Należy się w ogólności spodziewać, że zastosowanie w kodeku prostych algorytmów będzie się wiązać z niewielką złożonością obliczeniową techniki kodowania, jednak jej efektywność będzie co najwyżej umiarkowana. Zastosowanie algorytmów bardziej skomplikowanych zwiększy efektywność kompresji danych, jednak wzrost ten okupiony będzie dłuższym czasem kodowania i dekodowania danych. Ale jakie są konkretne wartości określające efektywność kompresji oraz obliczeniową złożoność współczesnych (rok 2018), zaawansowanych technik entropijnego kodowania danych? Jaka jest zależność wymienionych parametrów dla technik kodowania bazujących odpowiednio na algorytmie Huffmana oraz metodzie kodowania arytmetycznego?

Żeby odpowiedzieć na postawione pytania autor przeprowadził badania mające na celu wyznaczenie efektywności oraz złożoności technik kodowania entropijnego:

1. Universal Variable Length Coding (UVLC) [Richa03, AVC], stanowiącej połączenie metod kodowania:
    a. Kontekstowe adaptacyjne kodowanie o zmiennej długości słowa kodowego (CAVLC – Context Adaptive Variable Lenght Coding);
    b. Wykładnicze kodowanie Golomba (kodowanie Exp-Golomb-a);
2. Kontekstowe Adaptacyjne Binarne Kodowanie Arytmetyczne (CABAC – Context-based Adaptive Binary Arithmetic Coding) [Marp03a, Marp04, Richa03, Karw08].

Wymienione techniki kodowania entropijnego są stosowane w koderze wizyjnym MPEG-4 AVC/H.264 (technika CABAC również w koderze wizyjnym HEVC) i stanowią niewątpliwie odzwierciedlenie aktualnego stanu nauki i techniki w zakresie entropijnej kompresji danych obrazowych (rok 2018)[31]. Stąd dokonany przez autora wybór tych właśnie technik kodowania danych do badań efektywności oraz złożoności współczesnych kodeków entropijnych.

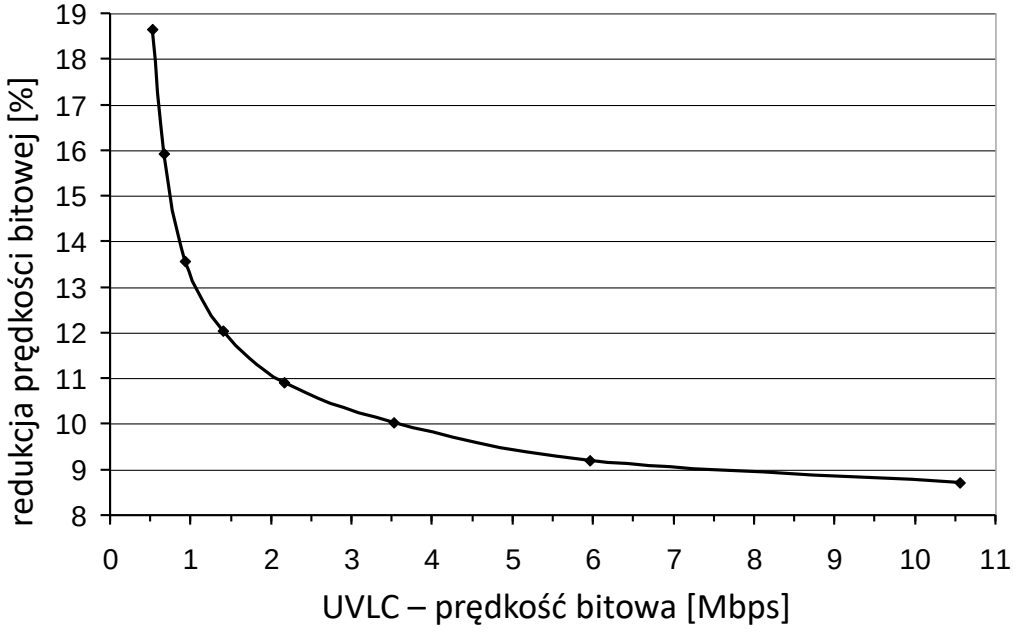
3.4.1. Efektywność kompresji algorytmów CABAC i UVLC

W koderach wizyjnych kodowanie entropijne stanowi ostatni etap kompresji danych obrazowych. Badania autora pokazują, że w przypadku kodera MPEG-4 AVC/H.264 zastosowanie kodowania entropijnego (CABAC lub UVLC) prowadzi do **3,5 – 7 krotnej** redukcji objętości

[31] Co prawda opracowano już w ostatnim czasie ulepszenia tych technik, jednak nie znalazły one szerokiego, praktycznego zastosowania w koderach wizyjnych.

danych w koderze, w porównaniu z sytuacją bez kodera entropijnego (w badaniach założono, że wyłączenie kodera entropijnego jest równoznaczne z koniecznością kodowania danych na stałej, z góry ustalonej liczbie bitów, wynikającej z zakresu dynamicznego kodowanych danych). Szczegółowe wyniki eksperymentu zależą od wielu czynników, jak choćby treści sekwencji, ustawień kodera, prędkości bitowej wynikowego strumienia danych, i zostały przedstawione w publikacji naukowej [Karw12]. Wyniki badań potwierdzają jednoznacznie duże znaczenie technik entropijnego kodowania danych w efektywnej reprezentacji danych multimedialnych.

Kodowanie arytmetyczne, które pozwala na przypisanie poszczególnym symbolom danych ułamkowej liczby bitów jest w praktyce znacznie bardziej efektywne od techniki kodowania Huffmana, która operuje na słowach kodowych o całkowitoliczbowej (a nie ułamkowej) liczbie bitów. Bazujący na kodowaniu arytmetycznym algorytm CABAC wytwarza w koderze AVC strumień bitowy, którego wielkość jest o **9-20% mniejsza** niż w przypadku zastosowania techniki UVLC. Dokładne wartości redukcji prędkości bitowej, wynikające z zastosowania algorytmu CABAC zamiast UVLC zależą od prędkości bitowej zakodowanego strumienia wizyjnego i zostały przedstawione na rysunku 3-17. Należy podkreślić, że w domenie kodowania entropijnego, które jest kodowaniem całkowicie bezstratnym, uzyskanie około 10% (oraz większej) redukcji wielkości strumienia bitowego to bardzo dużo. Tym bardziej, że w porównaniu tym odniesieniem była technika UVLC, która jest jedną z najbardziej efektywnych metod kodowania o zmiennej długości słowa kodowego. Stąd, obserwowany w ostatnim czasie, wzrost zainteresowania metodami kompresji bazującymi na arytmetycznym kodowaniu danych.



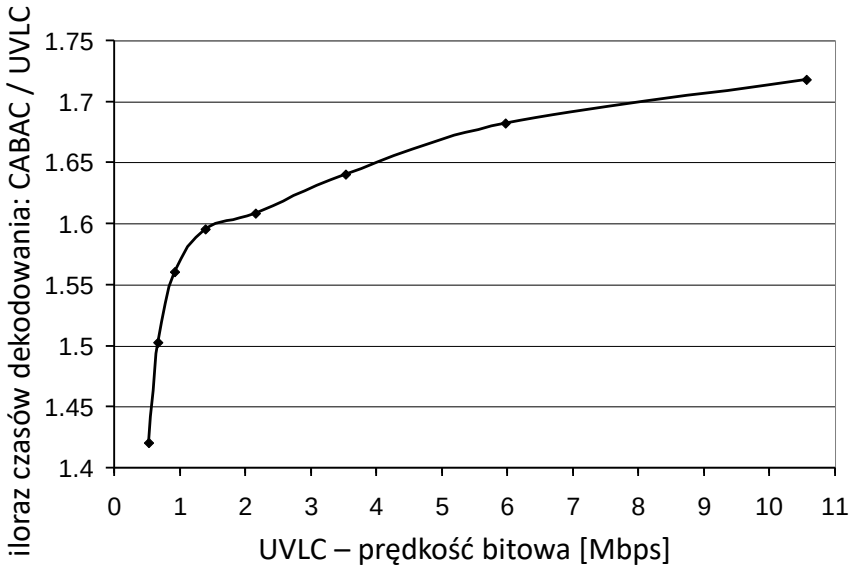
Rysunek 3-17. Redukcja prędkości bitowej (wartość średnia) będąca wynikiem zastosowania w koderze wizyjnym algorytmu CABAC, zamiast algorytmu UVLC. Średnia z czterech sekwencji testowych: CITY, CREW, HARBOUR, ICE. Wyniki badań własnych autora.

3.4.2. Złożoność obliczeniowa technik CABAC i UVLC

Wysoka efektywność techniki CABAC jest uzyskiwana kosztem zwiększonej złożoności kodeka entropijnego. W praktyce, złożoność techniki CABAC jest znacznie większa niż złożoność algorytmu UVLC. Dokładne czasy kodowania i dekodowania symboli danych z użyciem wspomnianych technik kompresji istotnie zależą od rodzaju wykorzystanego oprogramowania, jak również szczegółów metodologii samego eksperymentu. W kontekście badań przeprowadzonych na platformie IA-32 (Intel Core 2 Duo E6600 2,4 GHz z 4MB pamięci podręcznej 2-go poziomu, 4GB pamięci RAM), z wykorzystaniem własnej, programowej implementacji algorytmicznie zoptymalizowanego dekodera MPEG-4 AVC/H.264[32] (implementacja w języku programowania C), zdekodowanie jednego binarnego symbolu danych wymagało średnio[33]:

- Dla dekodera UVLC: **90 taktów procesora**;
- Dla dekodera CABAC: **160 taktów procesora**.

Z otrzymanych danych wynika, że niemała część obliczeń w dekoderze wizyjnym przypada na entropijne dekodowanie danych. W przypadku użytego w badaniach oprogramowania dekodera AVC dekodowanie CABAC stanowiło **10 – 20%** czasu dekodowania sekwencji. Obliczeniowa złożoność dekodera CABAC jest około **1,8 razy większa** niż złożoność dekodera UVLC. Na dokładne czasy entropijnego dekodowania symboli danych, jak również relację złożoności technik UVLC i CABAC wpływ ma treść kodowanej sekwencji oraz prędkość bitowa zakodowanego strumienia wizyjnego. Relację złożoności dekoderów UVLC i CABAC w funkcji prędkości bitowej strumienia danych przedstawiono na rysunku 3-18.



Rysunek 3-18. Porównanie złożoności dekoderów entropijnych: CABAC i UVLC. Uśrednione wartości na podstawie cząstkowych wyników uzyskanych dla sekwencji: CITY, CREW, HARBOUR, ICE. Wyniki badań własnych autora.

[32] Badania przeprowadzono dla strony dekodującej. Z algorytmicznego punktu widzenia złożoność kodera CABAC jest porównywalna do złożoności dekodera. W przypadku kodera UVLC, jego złożoność jest znacznie mniejsza niż złożoność dekodera.

[33] Badania przeprowadzone z wykorzystaniem zbioru sekwencji testowych o rozdzielczości przestrzennej 832x480, które zakodowano z docelową prędkością bitową w zakresie 1 – 20Mbps.

3.5. Podsumowanie

Niniejszy rozdział nie wyczerpuje tematyki entropijnej kompresji danych. Przedstawione w tym rozdziale techniki kodowania entropijnego stanowią zaledwie niewielką część znanych i opisanych w literaturze metod. W opisie ograniczono się do metod najbardziej popularnych, które znalazły szerokie zastosowanie w kompresji danych multimedialnych (obraz, dźwięk, i częściowo tekst). Dużo bardziej kompleksowe omówienie metod entropijnego kodowania danych znaleźć można w książkach [Salom06, Salom07, Salom10, Sayo00, Sayo12, Przel05].

Podstawą redukcji strumienia danych w koderze entropijnym jest uwzględnienie wiedzy o statystyce danych poddawanych kompresji. W przypadku danych, których statystyka podlega ciągłym zmianom (a tak jest w przypadku danych reprezentujących obraz czy dźwięk) trafne określenie prawdopodobieństw poszczególnych symboli danych jest bardzo trudne. Tym samym, sposób estymacji w koderze i dekoderze statystyki danych (a co się z tym wiąże, dokładność wyliczonych prawdopodobieństw symboli) rzutuje na efektywność całej metody kompresji danych. Chociaż problem ten został przez autora zasygnalizowany w punktach 3.2.3 oraz 3.3.13, to jednak metody statystycznego modelowania danych nie zostały tutaj szerzej omówione. Opis wybranych metod estymacji statystyki danych, będących technikami ogólnego przeznaczenia, znaleźć można w książkach [Salom06, Salom07, Salom10, Sayo00, Sayo12, Przel05]. Należy jednak podkreślić, że aktualnie wzrasta znaczenie dedykowanych metod statystycznego modelowania danych, które w coraz większym stopniu uwzględniają specyfikę (charakter) danych poddawanych kompresji (np. patrz sposób estymacji statystyki w algorytmie CABAC w koderach wizyjnych MPEG-4 AVC/H.264 oraz MPEG-H HEVC/H.265). Z tego względu, opracowanych dla konkretnych zastosowań, dedykowanych technik kompresji entropijnej nie można wprost zastosować do kodowania danych innego typu niż przewiduje to metoda kompresji.

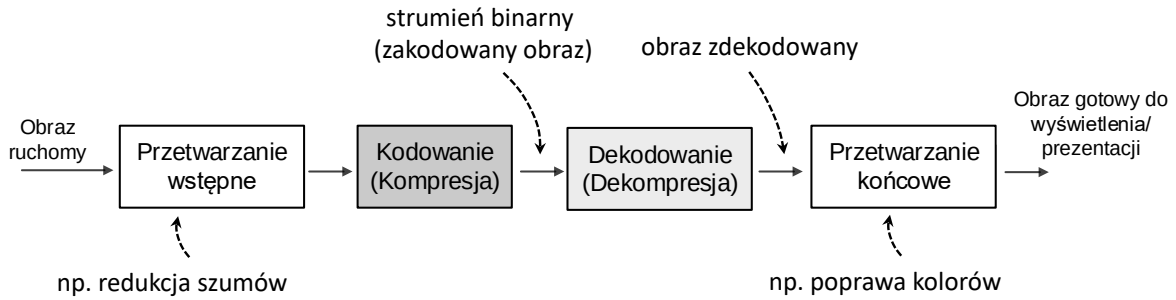
Rozdział 4

Wybrane techniki kompresji obrazu

4.1. Wprowadzenie

Reprezentacja obrazu w jego oryginalnej, nieskompresowanej formie wymaga ogromnej wręcz ilości danych. Standardowy sposób reprezentacji pojedynczego obrazu kolorowego o wysokiej rozdzielczości 1920x1080 próbek (8 bitów na próbkę każdej z trzech składowych obrazu kolorowego, czyli w sumie 24 bity na jedną „kolorową” próbkę) prowadzi do niemal 50 Mb (megabitów) danych. Typowy obraz ruchomy w tej rozdzielczości może zawierać 50 takich obrazów w ciągu każdej sekundy materiału, co tworzy nam w sumie prawie 2,5 Gb (gigabitów) danych informacyjnych w czasie jednej tylko sekundy! Jest więc oczywiste, że transmisja czy magazynowanie tak potężnego zbioru danych byłaby bardzo kosztowna, a w przypadku niektórych zastosowań, np. mobilne systemy multimedialne, po prostu niemożliwa. Problem ten tylko się potęguje wraz ze wzrostem przestrzennej rozdzielczości obrazów (np. telewizja 4K czy 8K) oraz wraz ze wzrostem liczby obrazów w ciągu jednej sekundy sekwencji (np. 120 obrazów na sekundę).

Właściwa transmisja obrazu, czy operacja jego zapisania na nośniku danych musi być zatem poprzedzona operacją redukcji tego ogromnego zbioru danych oryginalnych. Konieczne jest zastosowanie innego sposobu przedstawienia informacji o obrazie, który prowadziłby w efekcie do znacznie mniejszej liczby bitów reprezentujących obraz (w stosunku do jego reprezentacji oryginalnej). Wspomniany cel jest uzyskiwany w drodze **kompresji (kodowania)** oryginalnych **danych obrazowych**. Kompresja obrazu jest zatem zorientowana na redukcję rozmiaru strumienia bitowego, który opisuje nasz obraz. Wynikiem kompresji jest nowy strumień danych o znacząco mniejszym niż wcześniej rozmiarze. „Patrząc” na ten nowy strumień danych nie jesteśmy jednak w stanie „zobaczyć” wprost treści obrazu, którą on opisuje (reprezentuje). Żeby w tym nowym strumieniu danych otrzymać dostęp do wartości próbek obrazu i tym samym zobaczyć treść, którą przenosi obraz, konieczne jest przeprowadzenie **dekodowania** tegoż strumienia. Kompresję obrazu realizuje program lub urządzenie, które nazywamy **koderem**. Dekodowanie (dekompresja) to zadanie **dekodera** obrazu. W praktyce kompresję i dekompresję obrazu uzupełniają jeszcze operacje przetwarzania wstępnego (np. redukcja szumów w obrazie) oraz końcowego obrazów (np. poprawa kolorów), co prowadzi do całościowego toru przetwarzania obrazu, jak pokazano to na rysunku 4-1.



Rysunek 4-1. Ścieżka przetwarzania obrazu.

Kompresja danych, w tym danych, które opisują obrazy scen naturalnych, stanowi przedmiot bardzo intensywnych prac naukowych od kilkudziesięciu już lat. W toku tych badań dowiedziono istnienia w obrazie przesłanek (istnienia rodzajów nadmiarowości informacji w obrazie), dzięki którym możliwe są dużo wydajniejsze niż w przypadku reprezentacji oryginalnej, sposoby opisu treści obrazu. Tymi przesłankami są:

1. **Nadmiarowość statystyczna** danych obrazu. Poszczególne symbole danych, które reprezentują obraz, pojawiają się z różnym prawdopodobieństwem. Ten fakt stoi u podstaw **metod kodowania statystycznego** (inaczej metod **kodowania entropijnego**) danych obrazowych.

2. **Nadmiarowość przestrzenna** danych obrazu. W obrazie naturalnym sąsiednie fragmenty obrazu (czy sąsiednie próbki obrazu) wykazują bardzo silne podobieństwo. Zamiast kodować poszczególne fragmenty czy próbki obrazu zupełnie niezależnie od siebie, można wydajnie przewidywać treść tych fragmentów (czy wartości próbek) na podstawie danych sąsiednich i przesyłać do dekodera tylko błąd tego przewidywania. Obserwacja ta jest podstawą **metod kodowania predykcyjnego** danych obrazu.

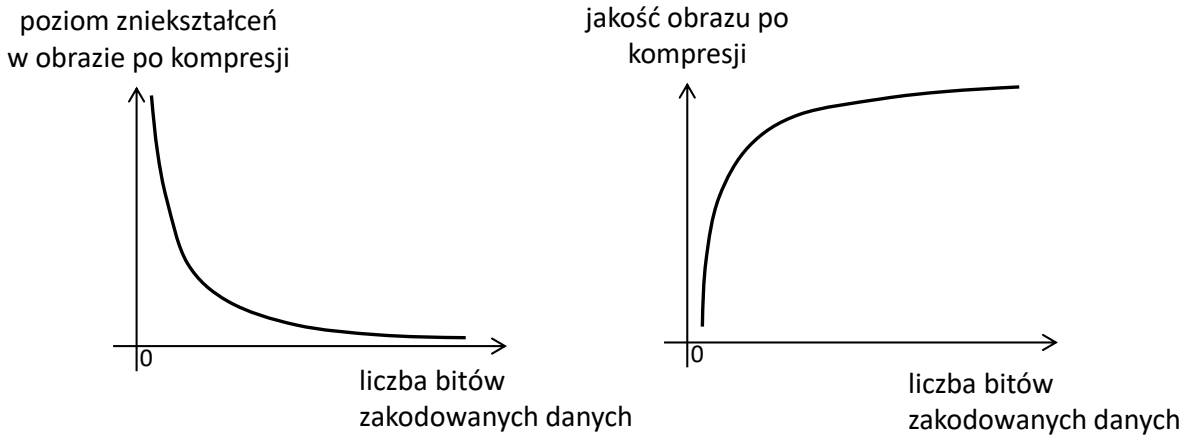
3. **Nadmiarowość czasowa**, w przypadku ruchomego obrazu. Jeśli rejestrująca scenę kamera nagrywa np. 50 obrazów w ciągu jednej tylko sekundy to znaczy to tyle, iż sąsiednie obrazy wykazują bardzo, ale to bardzo silne podobieństwo. Analogicznie więc, jak w przypadku nadmiarowości przestrzennej, zamiast odrębnego kodowania kolejnych obrazów dużo lepszym rozwiązaniem jest takie kodowanie, w którym treść bieżącego obrazu jest przewidywana na podstawie treści innych, sąsiednich obrazów. I w tym przypadku kodowaniu podlega sam błąd przewidywania treści obrazu. Mamy tutaj zatem ponownie odwołanie do **techniki kodowania predykcyjnego** obrazu.

4. **Nadmiarowość percepcyjna**. Rejestrujący obraz system widzenia człowieka nie działa w sposób idealny. Posiada on swoje ograniczenia, niedoskonałości. Z tego faktu wynika ważna obserwacja, iż nie wszystkie dane, które opisują obraz mają takie samo znaczenie z punktu widzenia percepcji obrazu. Część z tych danych, które uważamy za percepcyjnie nieistotne, możemy więc pominąć w trakcie kodowania lub zakodować je z mniejszą dokładnością. Osiągniemy w ten sposób znaczące oszczędności bitowe. Taka koncepcja kodowania prowadzi jednak do nieodwracalnej utraty części informacji o obrazie, co może skutkować widocznym dla odbiorcy spadkiem jakości obrazu.

Realizując kodowanie obrazu zgodnie z przesłankami 1-3 otrzymujemy gwarancję wiernego, czyli całkowicie bezstratnego odtworzenia oryginalnych danych obrazu, na podstawie danych po kompresji. Mówimy więc tutaj o całkowicie **bezstratnej kompresji danych**. Inaczej jest w przypadku realizowanego w drodze przesłanki 4 **kodowania stratnego** (nieodwracalnego) danych – w tym przypadku pominięcie podczas kodowania pewnych danych będzie oczywiście skutkować tym, że obraz odtworzony w dekoderze nie będzie tożsamy z obrazem oryginalnym. Jakość subiektywna tego odtworzonego obrazu może być znacząco niższa w porównaniu z jakością oryginalnego obrazu.

Poszczególne metody kodowania obrazów cechują się różną **efektywnością kompresji**. W przypadku metod kodowania bezstratnego efektywność algorytmu kompresji jest miarą jego zdolności do redukcji rozmiaru strumienia bitowego, który reprezentuje obraz. Zdolność ta jest wyrażana poprzez **współczynnik kompresji** (**stopień kompresji**) danych, który jest stosunkiem rozmiaru danych oryginalnych i rozmiaru danych po zakodowaniu. Metoda kodowania, która silnie redukuje strumień danych obrazu, cechuje się zatem wysokim współczynnikiem kompresji danych.

Metody **kompresji stratnej** obrazów wprowadzają nieodwracalne zmiany w ich treści. Przy ocenie efektywności tych metod, oprócz samego współczynnika kompresji danych, należy zatem dodatkowo uwzględnić również jakość obrazu odtwarzanego w dekoderze. W przypadku tych metod wyższa efektywność kompresji oznacza zdolność do uzyskania mniejszego rozmiaru strumienia zakodowanych danych przy danej jakości zakodowanego obrazu bądź równoważnie, zdolność do uzyskania obrazu zakodowanego o lepszej jakości przy danym rozmiarze strumienia zakodowanych danych. Efektywność techniki kompresji stratnej przedstawia się więc za pomocą wykresu, który prezentuje zależność pomiędzy **poziomem zniekształceń** w obrazie po kompresji, a **liczbą bitów** obrazu po zakodowaniu lub związek między **jakością obrazu** po kompresji, a **liczbą bitów** zakodowanego obrazu (patrz wykresy rysunku 4-2). Zgodnie z logiką, kodując obraz na małej liczbie bitów, należy się spodziewać wysokiego poziomu zniekształceń treści obrazu, czyli niskiej jakości zakodowanego obrazu. Jeśli tych bitów przeznaczymy dużo, to poziom zniekształceń będzie niski, czyli jakość obrazu po kompresji będzie wysoka.



Rysunek 4-2. Dwa sposoby (rysunek lewy i rysunek prawy) przedstawienia efektywności techniki kompresji stratnej obrazu.

Poziom zniekształceń obrazu należy więc tutaj rozumieć jako odwrotność jego jakości. Poziom zniekształceń w obrazie można mierzyć, sprawdzając na przykład, jaka jest różnica pomiędzy wartościami próbek obrazu przed i po kompresji. Jeśli ta różnica jest duża, to znaczy, że

poziom zniekształceń treści jest wysoki. Samą jakość obrazu można badać, wykonując **test subiektywny** (widz oglądając obraz, sam ocenia jego jakość) lub alternatywnie, można do tego celu użyć pewnej formuły matematycznej, która porównując obrazy przed i po kompresji „powie”, jaka jest jakość zakodowanego obrazu. W tym drugim przypadku mówimy o **obiektywnej ocenie** jakości obrazu. W stratnej kompresji obrazu najczęściej stosuje się w tym miejscu wskaźnik **PSNR** (ang. peak-signal to noise ratio) [Doma10]. Wskaźnik ten wyraża, w dziedzinie logarytmicznej, stosunek pomiędzy maksymalną możliwą energią sygnału (tym sygnałem jest obraz), a energią szumu, który zniekształca obraz (szum jest wyrażany jako kwadrat różnicy pomiędzy obrazem oryginalnym i zakodowanym).

Wymienione w tym punkcie przesłanki dla wydajnej reprezentacji obrazów są od wielu już lat powszechnie wykorzystywane w praktyce kompresji danych obrazowych. Przytoczone nadmiarowości danych obrazu stały się bodźcem do opracowania właściwych narzędzi kompresji danych, których zadaniem jest redukcja wspomnianych nadmiarowości. Tematem tego rozdziału są narzędzia kompresji danych obrazu, które znajdują szerokie, praktyczne zastosowanie w koderach obrazu.

Część I – Kompresja statycznego obrazu

4.2. Obraz kolorowy w przestrzeni YC_BC_R – czyli wstęp do kompresji

Rejestrowany przez kamerę (czy aparat) kolorowy obraz jest oryginalnie reprezentowany przy pomocy trzech składowych barwnych: **R**, **G** i **B**. Z uwagi jednak na silne podobieństwo tych trzech obrazów składowych (których niezależne kodowanie wprowadzałoby istotną nadmiarowość reprezentacji danych obrazu kolorowego) właściwa kompresja obrazu nie jest realizowana w oryginalnej przestrzeni **RGB**. Kompresję tę poprzedza wykonanie matematycznego przekształcenia, które w oparciu o składowe **R**, **G** i **B** wyznacza nowe składowe obrazu, oznaczane jako **Y**, $\mathbf{C_B}$, $\mathbf{C_R}$. Właśnie obrazy tych nowych składowych są przedmiotem właściwej kompresji.

Argumenty, które w pełni uzasadniają słuszność takiego podejścia zostały już przedstawione w punktach 1.4.5, 1.4.6 oraz 1.4.7. Jednak dla zachowania spójności tego fragmentu tekstu najważniejsze motywacje zostaną w tym miejscu raz jeszcze powtórzone:

1. Wspomniane przekształcenie matematyczne (przejścia z RGB na YC_BC_R) dekoreluje obrazy składowych **R**, **G** i **B**, powodując, że nowe składowe **Y**, $\mathbf{C_B}$, $\mathbf{C_R}$ nie wykazują już tak silnego podobieństwa (patrz poniższy rysunek). Niezależne kodowanie nowych składowych nie wprowadza więc nadmiarowości reprezentacji danych obrazu, która na pewno by wystąpiła w przypadku niezależnej kompresji składowych **R**, **G** i **B**. Łatwo ten wniosek potwierdzić wyznaczając wartość entropii *H* zarówno składowych **R**, **G** i **B**, jak również składowych **Y**, $\mathbf{C_B}$, $\mathbf{C_R}$. Wyniki tych obliczeń zostały zamieszczone na rysunku 4-3. Jak dobrze widać, entropia składowych koloru $\mathbf{C_B}$ i $\mathbf{C_R}$ jest znacznie mniejsza od entropii odpowiednio składowych **B** i **R** przestrzeni **RGB**, co przekłada się na istotnie mniejszy koszt reprezentacji składowych $\mathbf{C_B}$ i $\mathbf{C_R}$ (odniesienie do siebie wartości entropii właściwych składowych obrazu wskazuje na $\frac{7{,}5197}{5{,}1854} = 1{,}4502$ oraz $\frac{7{,}4998}{5{,}1064} = 1{,}4687$ krotną redukcję kosztu transmisji składowych $\mathbf{C_B}$ i $\mathbf{C_R}$ zamiast składowych **B** i **R**). [34]

[34] Koszt transmisji trzeciej składowej jest już porównywalny w obu przestrzeniach. Wskazane (graniczne) krotności redukcji kosztu reprezentacji składowych koloru dotyczą przypadku ich entropijnej kompresji.

2. W reprezentacji **Y**, $\mathbf{C_B}$, $\mathbf{C_R}$ mamy już wydzielone składowe (te dwie ostatnie), które przenoszą informację o samym tylko kolorze. Z uwagi na mniejszą wrażliwość aparatu widzenia człowieka na zmianę koloru w obrazie (w porównaniu do wrażliwości na zmianę jasności) informację o kolorze (czyli składowe $\mathbf{C_B}$ i $\mathbf{C_R}$) można kodować z mniejszą dokładnością niż dane składowej **Y** (informacja o jasności). Powszechną praktyką (w **stratnej kompresji** obrazu) jest stosowanie dla składowych koloru mniejszej częstotliwości próbkowania (w porównaniu z tą stosowaną dla składowej **Y**) oraz silniejszego kwantowania danych koloru. Zastosowanie samego tylko schematu próbkowania 4:2:0 składowych koloru aż 4-krotnie redukuje liczbę próbek składowych $\mathbf{C_B}$ i $\mathbf{C_R}$, które podlegają dalszej kompresji w koderze.

Oryginalne dane, które opisują poszczególne składowe przestrzeni $\mathbf{YC_BC_R}$, ciągle jednak wykazują bardzo dużą nadmiarowość (np. nadmiarowość przestrzenną, statystyczną itd.). Stosując odpowiednie metody kodowania tych danych można tę nadmiarowość istotnie zmniejszyć, skutkiem czego jest dalsza redukcja bitowego kosztu ich reprezentacji. Z tej perspektywy największym problemem jest wydajne kodowania składowej **Y** obrazu – dane, które wchodzą w skład tej składowej, są trudniejsze do zakodowania (w porównaniu z kodowaniem składowych koloru), przez co stanowią większą część danych, które reprezentują obraz. Dlatego też to wydajna kompresja właśnie składowej **Y** będzie dalej przedmiotem szczegółowych rozważań. Przedstawione dalej metody mogą być również użyte do wydajnej kompresji składowych koloru.

Składowe przestrzeni **RGB** w wersji monochromatycznej

składowa **R**

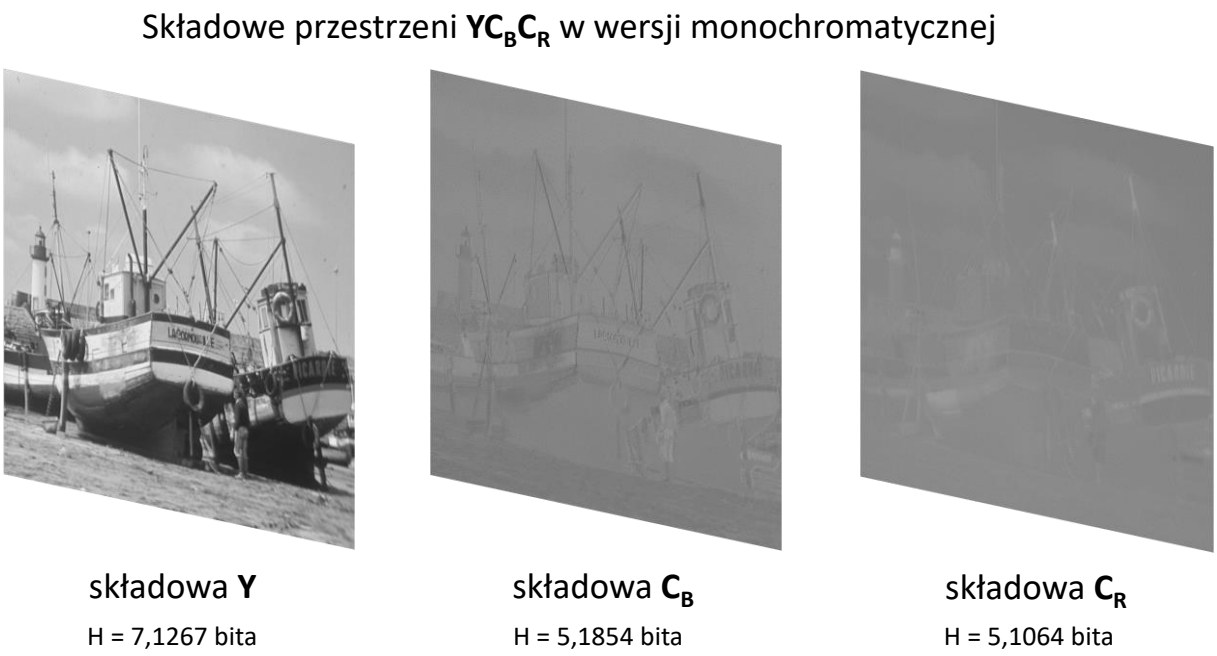
H = 7,4998 bita

składowa **G**

H = 7,2562 bita

składowa **B**

H = 7,5197 bita

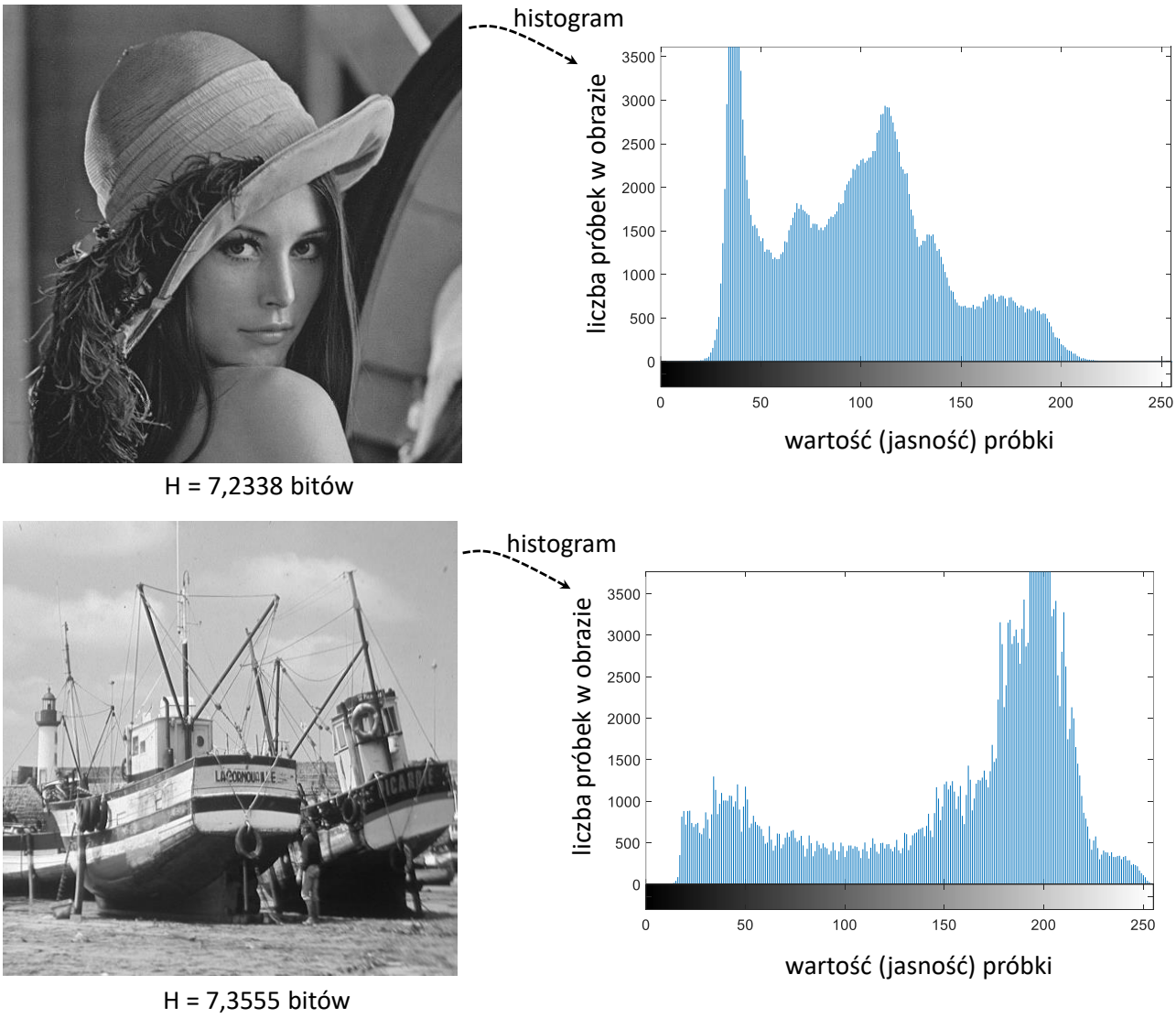


Rysunek 4-3. Składowe **R**, **G**, **B** oraz **Y**, C_B, C_R kolorowego obrazu „Boats”. Porównanie stopnia podobieństwa trzech składowych obrazu w reprezentacjach **RGB** i YC_BC_R.

4.3. Statystyczna nadmiarowość próbek obrazu – prosty sposób jej redukcji

Przypuszczalnie najprostszym sposobem zakodowania treści obrazu jest przeprowadzenie **entropijnego kodowania** wartości jego próbek. Jak już dobrze wiemy z lektury 3 rozdziału, metody kodowania entropijnego potrafią efektywnie zakodować symbole danych, jeśli prawdopodobieństwa występowania poszczególnych symboli wykazują duże zróżnicowanie. Duże, czyli takie, zgodnie z którym większość symboli danych pojawia się z bardzo małym prawdopodobieństwem, i tylko w przypadku (bardzo) małej liczby symboli prawdopodobieństwa ich wystąpienia są większe.

Tak jednak niestety nie jest w przypadku symboli danych, będących próbkami obrazu. Dobrze to widać analizując wykresy **histogramów** obrazu, które pokazują ile w obrazie jest próbek o danej wartości, co odpowiada właśnie informacji o statystycznym rozkładzie wartości próbek w obrazie. Histogramy dla przykładowych monochromatycznych obrazów testowych „Lena” oraz „Boats” zostały przedstawione na poniższym rysunku. Każda próbka obrazów testowych jest reprezentowana na 8 bitach.



Rysunek 4-4. Obrazy testowe „Lena” oraz „Boats” oraz ich histogramy. Każda z próbek obrazu jest zapisana na 8 bitach (co daje 256 możliwych wartości próbek). Wartość entropii próbek przedstawionych obrazów wynosi odpowiednio $H = 7{,}2338$ bitów i $H = 7{,}3555$ bitów.

Analiza tych histogramów prowadzi nas do wniosku, że chociaż częstości wystąpienia próbek o danej wartości nie są w ogólności takie same, to jednak w przypadku (relatywnie) szerokiego zakresu wartości próbek prawdopodobieństwa ich występowania w obrazie są do siebie bardzo zbliżone. Można więc powiedzieć, że histogramy te są w znacznym stopniu wyrównane, co jest właśnie cechą charakterystyczną obrazów scen naturalnych o dobrym kontraście. W takich obrazach zróżnicowanie statystyki wartości próbek na pewno nie jest duże.

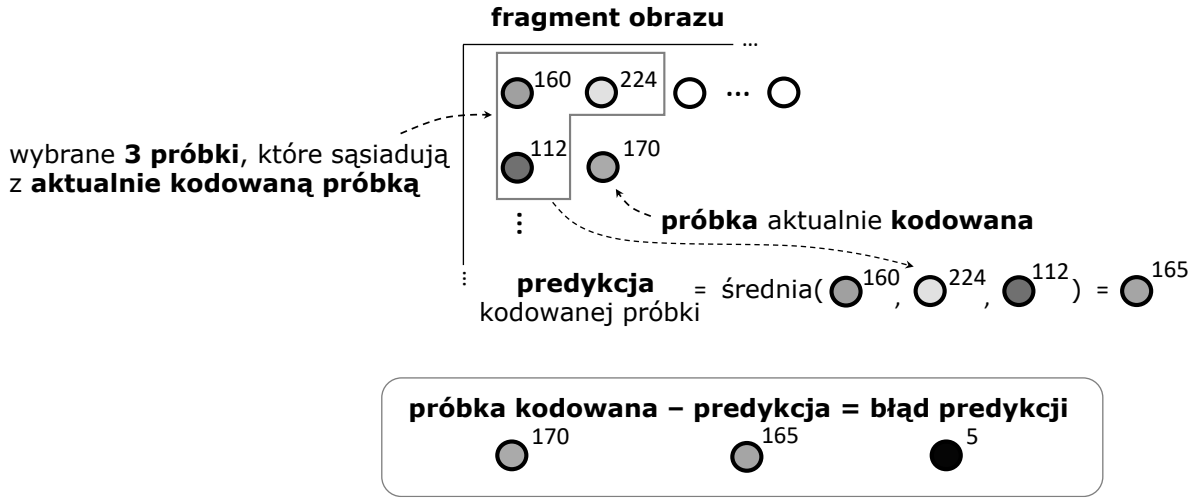
W związku z powyższą obserwacją entropijne kodowanie wartości próbek obrazu będzie zwykle skutkować słabymi efektami redukcji strumienia danych, który opisuje obraz. Łatwo się o tym przekonać obliczając entropię wartości próbek obrazu, która określa granicę możliwości (efektywności) entropijnej kompresji danych. Dla przedstawionych dwóch obrazów testowych („Lena” i „Boats”) entropia ta wynosi odpowiednio $H = 7{,}2338$ bitów i $H = 7{,}3555$ bitów. Oznacza to tyle, że realizując entropijne kodowanie próbek obrazów „Lena” i „Boats” reprezentacja pojedynczej próbki tych obrazów wymagałaby średnio co najmniej $7{,}2338$ bitów

(dla obrazu „Lena”) oraz $7{,}3555$ bitów (w przypadku obrazu „Boats”). Co najmniej, czyli są to efektywności graniczne kodowania entropijnego, czyli najlepsze z możliwych w przypadku entropijnego kodowania tych konkretnych danych. Porównując te rezultaty do kosztu 8 bitów opisania jednej próbki w przypadku oryginalnej reprezentacji obrazów (czyli bez kodowania entropijnego) widać, że wartości przywołanych entropii są duże. Stopień kompresji entropijnej wynosi więc zaledwie $\frac{8}{7{,}2338} = 1{,}1059$ oraz $\frac{8}{7{,}3555} = 1{,}0876$, odpowiednio dla obrazów „Lena” i „Boats”, co przekłada się na bardzo niewielką, bo około 10% i 9% redukcję strumieni danych obrazów. Dlatego w praktyce, kompresja obrazu nie bazuje na samym tylko entropijnym kodowaniu jego próbek.

4.4. Przewidywanie danych obrazu jako sposób na poprawę wyników kompresji

Osiągnięta w poprzednim punkcie bardzo niska efektywność kompresji danych obrazu była spowodowana tym, iż kolejne próbki obrazu kodowane były zupełnie niezależnie od siebie. W zastosowanym prostym mechanizmie entropijnej kompresji próbek obrazu w ogóle nie wykorzystano faktu istnienia **przestrzennej nadmiarowości danych obrazu**, czyli faktu silnego podobieństwa wartości sąsiadujących ze sobą próbek. Można powiedzieć, że w drodze niezależnej kompresji próbek niejako wielokrotnie kodowaliśmy pewną część informacji, która powtarzała się w kolejnych, położonych blisko siebie, próbkach. W efekcie otrzymaliśmy zatem bitową reprezentację danych obrazu, która ciągle wykazywała znaczącą nadmiarowość.

Jednak osiągnięte w poprzednim punkcie słabe rezultaty kompresji można w dość prosty sposób poprawić. Skoro sąsiednie próbki w obrazie mają podobne do siebie wartości, to naturalnym rozwiązaniem jest zastosowanie w koderze obrazu mechanizmu **przewidywania,** czyli **predykcji** wartości kolejnych próbek obrazu, w oparciu o próbki, które z nimi sąsiadują. Można na przykład przyjąć, że dobrą estymatą (szacunkiem) wartości aktualnie kodowanej próbki obrazu będzie średnia z wartości trzech próbek, które bezpośrednio sąsiadują z próbką kodowaną: tzw. próbka lewa, górna, oraz górna-lewa, jak zostało to zaznaczone na poniższym rysunku.



Rysunek 4-5. Ilustracja idei przewidywania wartości kolejnych próbek obrazu na podstawie znajomości wartości próbek, które sąsiadują z próbkami kodowanymi.

Oczywiście, nie w każdym przypadku wartość aktualnie kodowanej próbki obrazu będzie dokładnie równa średniej wartości jej trzech sąsiadów. Zatem w ogólności, w drodze przewidywania wartości próbek możemy popełnić pewien błąd, pomyłkę, dodatnią bądź ujemną, która jest określana mianem **błędu przewidywania**, czy **błędu predykcji** wartości próbek. Z uwagi jednak na wspomnianą już wcześniej silną korelację (podobieństwo) wartości sąsiadujących ze sobą próbek pomyłka ta (czyli błąd predykcji) będzie zwykle przyjmować zerowe bądź bardzo małe wartości, dodatnie czy ujemne. Duże wartości błędu predykcji próbek (dodatnie bądź ujemne) mogą się oczywiście również pojawić, jednak jak wskazuje na to cecha dużego podobieństwa sąsiednich fragmentów obrazu będzie to sytuacja dość rzadka.

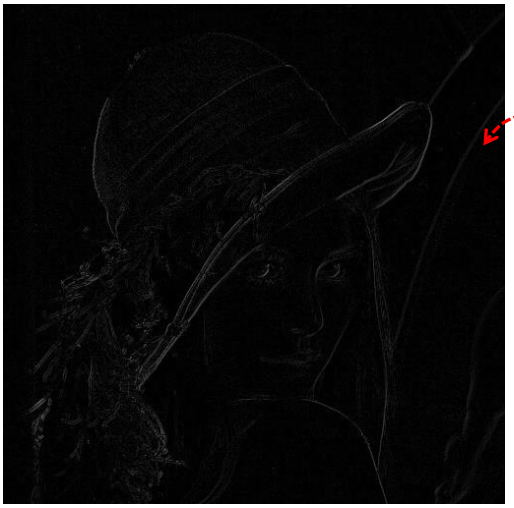
Przytoczone wnioski możemy łatwo zaobserwować, przyglądając się zamieszczonym na rysunku 4-6 obrazom błędu predykcji próbek oraz rozkładom wartości tych błędów, które otrzymano dla przykładowych obrazów testowych „Lena” oraz „Boats”. W pierwszej kolejności, przy pomocy odcieni szarości (które są w obrazie wyrażane całkowitymi liczbami z zakresu od 0 do 255), pokazano jak przedstawia się moduł (czyli wartość bezwzględna) błędu predykcji próbek. Ten sposób prezentacji charakteru sygnału błędu przewidywania wartości próbek nie uwzględnia jednak istotnej informacji o znaku tego błędu. Dlatego w drugiej kolejności, celem możliwości zobrazowania zarówno dodatnich, jak i ujemnych wartości próbek błędu predykcji, do każdej próbki błędu została dodana stała wartość 128. W obrazach przedstawiających moduł błędu predykcji próbek błąd zerowy jest przedstawiony kolorem czarnym, natomiast inne wartości błędu odpowiednio wyższymi poziomami jasności (aż do koloru białego). Z kolei obrazy, które prezentują błąd predykcji próbek, wraz z informacją o znaku tego błędu, przedstawiają zerowy błąd kolorem szarym (wartość 128 na histogramach), natomiast duże błędy predykcji, dodatnie bądź ujemne, odpowiednio odcieniami kolorów białego bądź czarnego.

Jak więc dobrze widać, w przedstawionych obrazach błędu predykcji próbek zdecydowanie przeważają obszary, które odpowiadają zerowej, bądź innej, bardzo małej (dodatniej bądź ujemnej) wartości błędu. Tylko w tych bardziej złożonych fragmentach obrazu, które zawierają krawędzie, czy inny typ złożonej tekstury, popełniony został większy błąd przewidywania wartości próbek, jednak w skali całego obrazu takich sytuacji było relatywnie mało. W konsekwencji wynikowe obrazy błędu predykcji próbek przenoszą bardzo prosty sygnał resztkowy predykcji (czyli błąd predykcji), sygnał, który jest znacznie mniej złożony w porównaniu z sygnałem, który opisuje oryginalną treść obrazów. Dobrze to widać zestawiając ze sobą obrazy błędu predykcji z obrazami oryginalnymi. Ten dużo prostszy sygnał resztkowy predykcji przełoży się na mniejszy niż w przypadku kodowania oryginalnego obrazu, koszt bitowy jego reprezentacji.

Żeby powyższy wniosek jednoznacznie potwierdzić, dokonajmy entropijnej kompresji wartości błędu predykcji próbek. Jeszcze przed wykonaniem tej operacji, analizując szczegółowo postać histogramu próbek błędu, widzimy jak silne jest zróżnicowanie częstości wystąpienia błędu predykcji o danej wartości (z plusem lub z minusem). Przypomnijmy, że takiego zróżnicowania nie obserwowaliśmy na histogramach, które zostały wyznaczone dla próbek obrazów. Owe silne zróżnicowanie statystyki próbek błędu daje nam gwarancję tego, że kodowanie entropijne tych próbek da w rezultacie wyniki kompresji znacznie lepsze niż miało to miejsce, kiedy kodowaniu podlegały bezpośrednio próbki obrazu. I faktycznie, entropia próbek błędu predykcji jest w tym przypadku znacznie mniejsza i wynosi $H = 4{,}8813$ bita i $H = 5{,}3465$ bita, odpowiednio dla wyników przewidywania obrazów „Lena” i „Boats” (porównaj z przedstawionymi w poprzednim punkcie wynikami dla próbek). Entropijna kompresja błędu predykcji próbek pozwoliłaby więc osiągnąć w najlepszym razie $1{,}6389$ oraz $1{,}4963$ – krotną redukcję strumieni danych, które opisywałyby obrazy „Lena” i „Boats” (przytoczone krotności określają stopnie kompresji obrazów

„Lena” i „Boats”: $\frac{8}{4{,}8813} = 1{,}6389$ oraz $\frac{8}{5{,}3465} = 1{,}4963$). W porównaniu z osiągniętym wcześniej (przy entropijnej kompresji próbek obrazu, a nie błędu przewidywania tych próbek) stopniem kompresji na poziomie $1{,}1$ jest to zatem dość znacząca poprawa efektywności kompresji danych obrazowych. Tę poprawę zawdzięczamy przeprowadzonej wcześniej wstępnej dekorelacji danych wejściowych, jeszcze przed użyciem techniki kompresji entropijnej. W naszym przypadku dekorelacja próbek obrazu doprowadziła do silnego ograniczenia obecnej w danych obrazu **nadmiarowości przestrzennej**.

„Lena”: moduł błędu predykcji próbek
(bez informacji o znaku błędu)

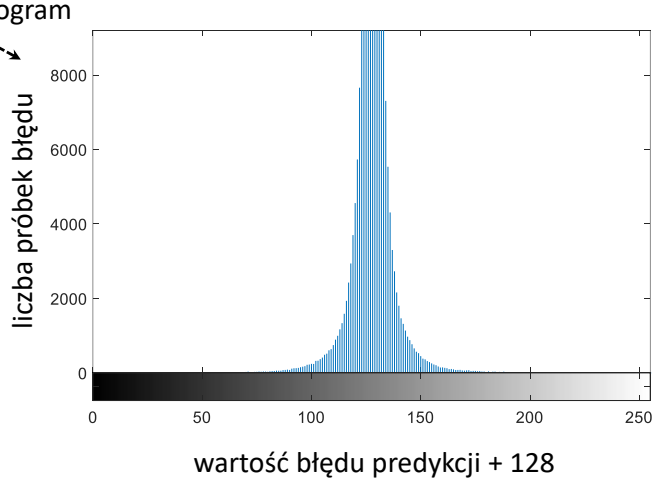


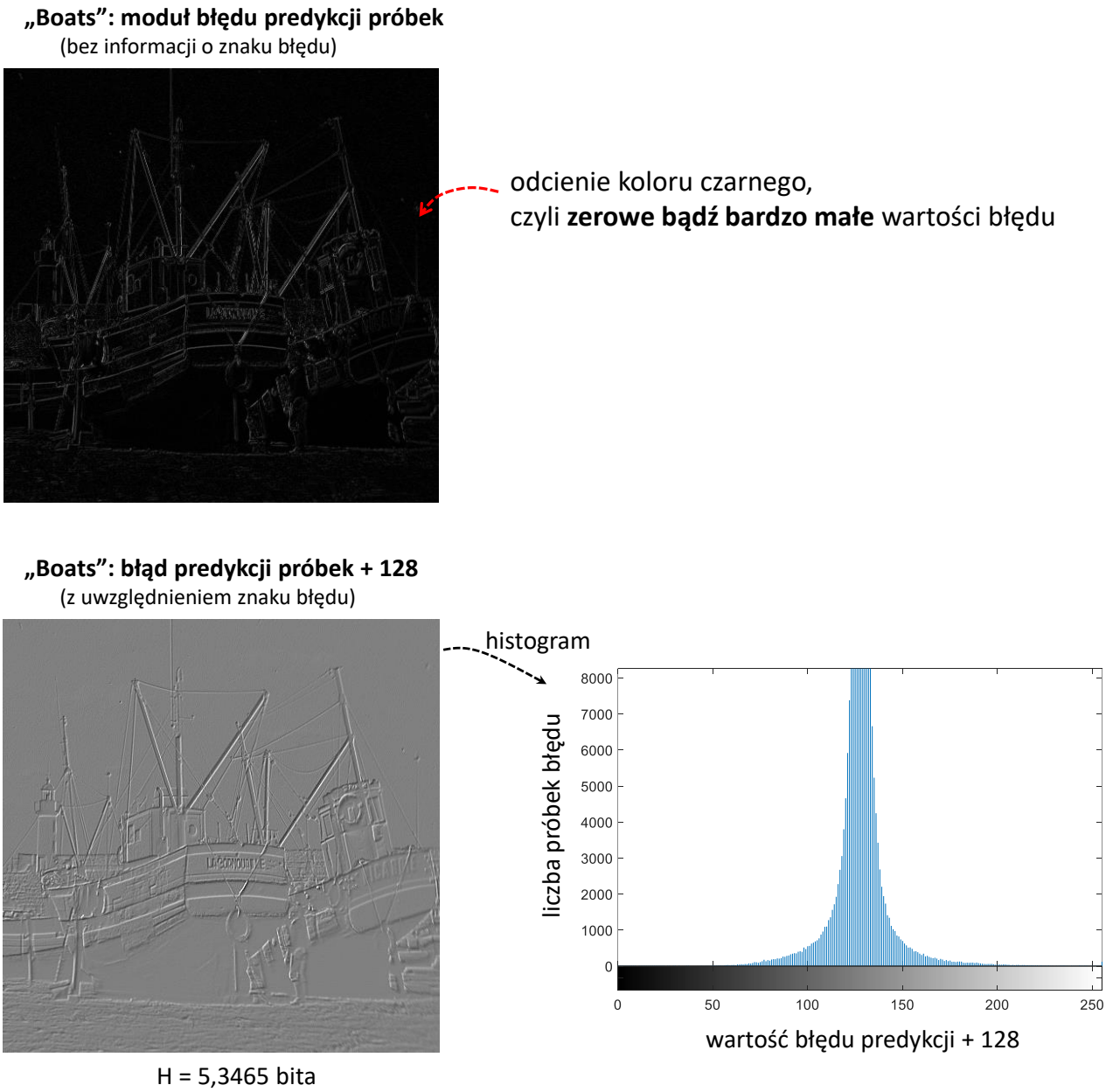
odcienie koloru czarnego,
czyli **zerowe bądź bardzo małe** wartości błędu

„Lena”: błąd predykcji próbek + 128
(z uwzględnieniem znaku błędu)

H = 4,8813 bita

histogram





Rysunek 4-6. Błędy predykcji (przewidywania) obrazów „Lena” i „Boats” oraz histogramy obrazów błędów predykcji. Obrazy przedstawiają odpowiednio moduł błędu predykcji próbek oraz błąd predykcji próbek wraz z informacją o znaku błędu. W tym drugim przypadku, do każdej próbki błędu została dodana stała wartość 128, celem możliwości zobrazowania wartości błędów poziomami jasności, które są reprezentowane liczbami całkowitymi z zakresu od 0 do 255. Wartość entropii próbek błędu predykcji dla przedstawionych obrazów wynosi odpowiednio $H = 4{,}8813$ bita i $H = 5{,}3465$ bita.

4.5. Przewidywanie wartości próbek obrazu – jak to się robi w praktyce?

W poprzednim punkcie przedstawiony został bardzo prosty sposób przewidywania wartości próbek obrazu. Wszystkie próbki przewidywane były zgodnie z tym samym, ustalonym wcześniej schematem uśrednienia wartości trzech próbek, które bezpośrednio sąsiadowały z

aktualnie przewidywaną próbką. Taki sposób predykcji próbek, w ogóle więc nie uwzględniał wiedzy o treści przewidywanego obrazu jako podstawy do określenia próbki lub zbioru próbek, z których najlepiej jest przewidywać wartość próbki bieżącej. Dlatego nie mógł on w pełni pokazać olbrzymiego potencjału, jaki się kryje za metodami kodowania predykcyjnego obrazu.

W najnowszych koderach obrazu (tych, które zostały opracowane po roku 2000) stosuje się dużo bardziej zaawansowane mechanizmy przewidywania próbek. Podstawą działania tych mechanizmów jest zastosowanie nie jednego, ustalonego na sztywno sposobu przewidywania wartości próbek, ale zbioru wielu różnych sposobów, które z dużą dokładnością potrafią przewidywać treści obrazu o różnym charakterze. Przywołane tutaj sposoby przewidywania wartości próbek powszechnie się nazywa **predyktorami wartości próbek** obrazu. Praktyka wyraźnie pokazuje, iż łączne wykorzystanie w koderze obrazu wielu różnych predyktorów, razem z wyborem najlepszego predyktora dla kolejnych fragmentów obrazu, pozwala uzyskać znacznie lepsze rezultaty przewidywania treści niż w przypadku użycia tylko jednego predyktora. Efektem jest oczywiście mniejszy niż w przypadku prostej metody predykcji próbek, błąd predykcji próbek, co przekłada się na mniejszy koszt bitowy reprezentacji zakodowanego w ten sposób obrazu.

Jednak dla osiągnięcia powyższego celu kluczowe jest właściwe określenie zbioru stosowanych przez koder predyktorów treści obrazu. Wybrane predyktory mają umożliwić dokładne przewidywanie różnego rodzaju treści, jakie mogą się pojawić w kodowanym obrazie. W obrazach scen naturalnych spotkać można:

- Jednolite fragmenty, w których wartości próbek nie podlegają szybkim zmianom bądź nie zmieniają się w ogóle. Takich fragmentów jest z reguły w obrazie najwięcej.
- Obszary bardziej od nich złożone, które zawierają np. krawędzie obrazu biegnące pod określonym kątem.
- Fragmenty obrazu, które zawierają obszary o płynnie (lub prawie płynnie) zmieniającej się wartości próbek (chodzi tutaj o obszary, które daje się opisać pewną liniową funkcją). Taką treść daje się odwzorować płaszczyzną, powierzchnią, w której następuje określona zmiana jasności próbek.

Z tej perspektywy naturalnym wyborem jest wykorzystanie następujących predyktorów treści:

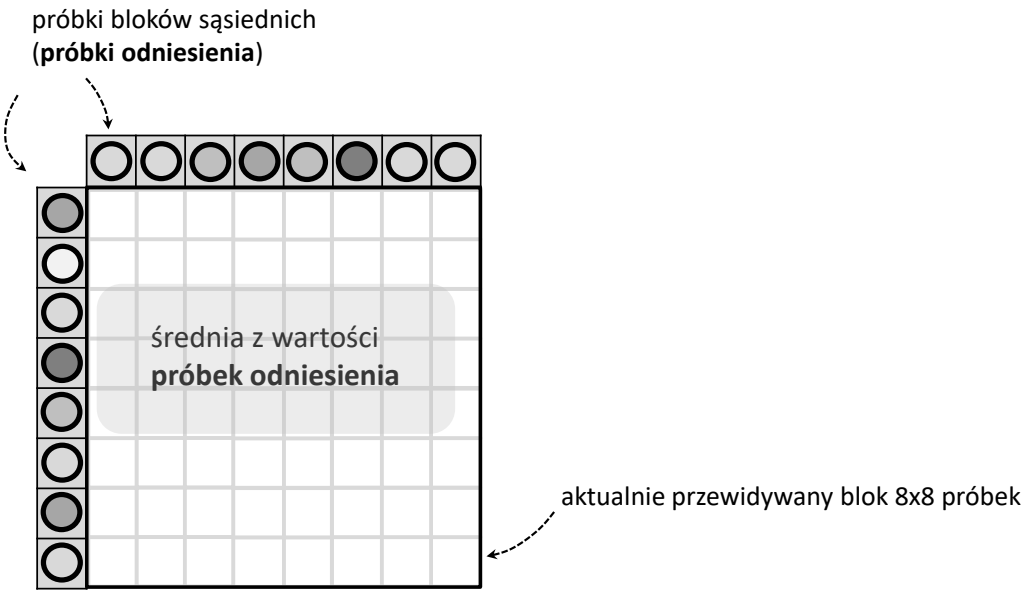
1) Predyktora wartości stałej, nazywanego dalej **predyktorem DC**. Potrafi on dobrze przewidywać mało skomplikowane treści obrazu, np. gładkie, jednolite obszary.

2) Pewnej liczby **predyktorów kątowych**, zdolnych do przewidywania fragmentów obrazu, w których występują krawędzie. Poszczególne predyktory z tej grupy umożliwiają wydajne przewidywanie krawędzi, które biegną w obrazie pod określonym kątem. Z punktu widzenia uzyskiwanych rezultatów kodowania predykcyjnego, zasadne jest użycie dużej liczby predyktorów kątowych (nawet więcej niż 30), które są w stanie obsłużyć szeroki zakres kątów.

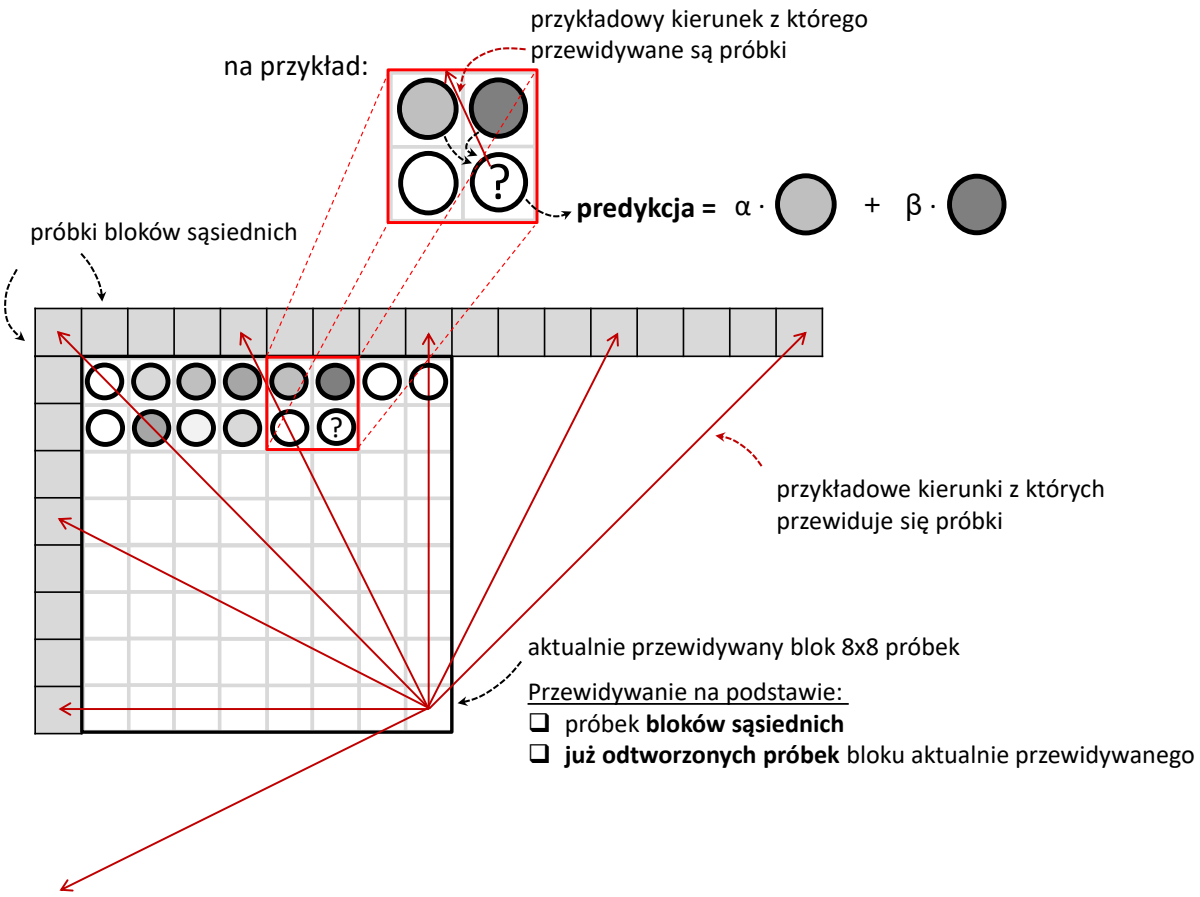
3) Predyktora, określanego w anglojęzycznej literaturze mianem **PLANE**, który jest zaplanowany do wydajnego przewidywania obszarów o liniowej, płynnie zmieniającej się jasności próbek. Działanie tego predyktora można rozumieć jako rozpostarcie (rozciągnięcie) odpowiedniej płaszczyzny, pomiędzy pochodzącymi z sąsiednich bloków obrazu próbkami odniesienia.

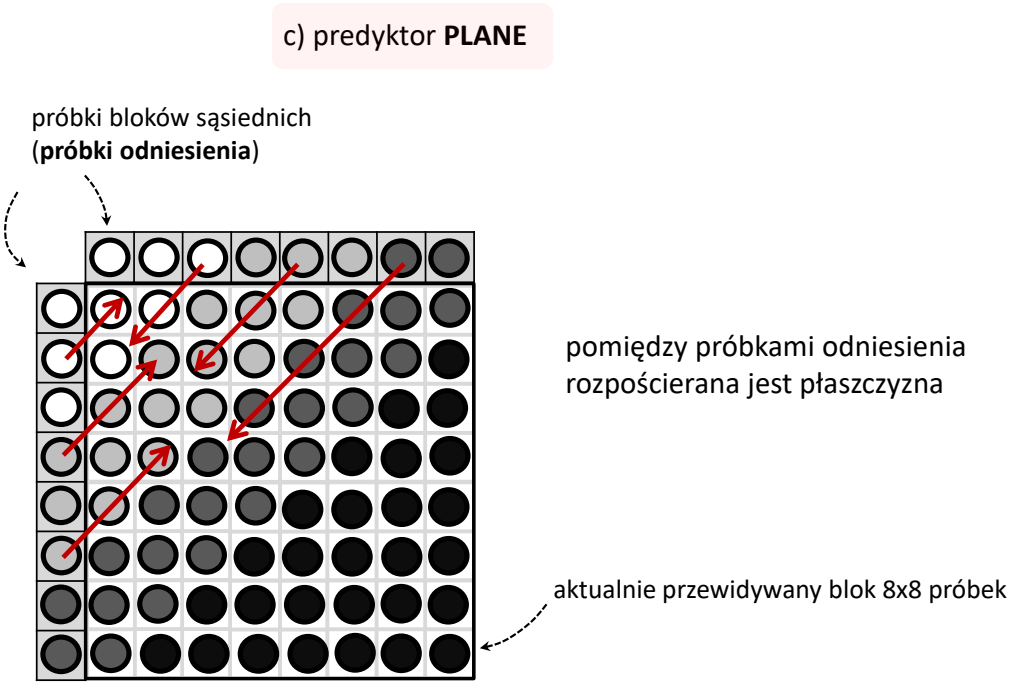
Otrzymany w ten sposób przykładowy zbiór predyktorów wartości próbek został przedstawiony na poniższym rysunku.

a) predyktor **DC**



b) predyktory **kątowe**





Rysunek 4-7. Przykładowy zbiór predyktorów wartości próbek, wraz z ilustracją idei działania poszczególnych predyktorów. Zbiór ten zwykle zawiera: predyktor DC (a), predyktory kątowe (b), oraz predyktor PLANE (c).

Łączne użycie przedstawionych sposobów predykcji prowadzi do bardzo dobrych rezultatów przewidywania treści obrazu. Wysoka skuteczność tych metod została dobrze potwierdzona licznymi badaniami, np. w testach wydajności technik kompresji AVC i HEVC, w których znajdują one praktyczne zastosowanie.

Użycie wielu dedykowanych predyktorów daje możliwość adaptacyjnego wyboru tego spośród nich, który z najwyższą skutecznością potrafi przewidzieć aktualnie przetwarzany fragment obrazu. Z teoretycznego punktu widzenia ten najlepszy predyktor powinien być wybierany dla każdej kolejnej próbki obrazu niezależnie. Wydaje się więc, że decyzja o sposobie predykcji wartości powinna być podejmowana możliwie często. W takim przypadku uzyskuje się najmniejszą (oczywiście dla danego zestawu predyktorów) pomyłkę przewidywania treści. Jednak przełączanie się pomiędzy dostępnymi sposobami predykcji wymaga jeszcze zastosowania odpowiedniej sygnalizacji użytego trybu – do dekodera koniecznie trzeba wysłać informację o tym, jaki tryb predykcji próbek został w danym momencie użyty. Transmisja do dekodera takiej informacji co próbkę wiązałaby się z tak dużym kosztem bitowym, że zniwelowałby on otrzymane wcześniej zyski z dokładniejszej predykcji próbek. Dlatego w praktyce, wyboru najlepszego trybu trzeba dokonywać znacznie rzadziej. Doświadczenie pokazuje, że bardzo dobrym kompromisem pomiędzy dokładnością, z jaką można przewidywać próbki, a kosztem przekazania dekoderowi informacji o zastosowanym trybie, jest jego wybór na poziomie bloków próbek, a nie pojedynczych próbek. W tym przypadku wszystkie próbki bloku są przewidywane z użyciem tego samego, wcześniej wybranego sposobu predykcji. Takie rozwiązanie również zapewnia wysoką skuteczność przewidywania treści bloku, przy czym silnie redukuje rozmiar pobocznej informacji, która daje dekoderowi wiedzę o zastosowanym trybie.

Przy takim podejściu kluczowy jest jeszcze rozmiar bloku, dla którego wybiera się sposób predykcji wartości jego próbek. Intuicja podpowiada, że również i tę decyzję najlepiej jest uzależnić

od charakteru treści, która znajduje się w przetwarzanym fragmencie obrazu. Jeśli jakiś fragment obrazu jest jednolitą teksturą, to nie warto dla niego stosować dużej liczby małych bloków i wybierać w nich niezależnie najlepszy predyktor. Takie rozwiązanie nie zwiększyłoby w ogólności skuteczności predykcji, ponieważ z bardzo dużym prawdopodobieństwem ten sam typ predyktora zostałby wybrany w każdym z małych bloków. Natomiast koszt sygnalizacji predyktorów, które zostały użyte w tych małych blokach, byłby z pewnością duży. Jednolite fragmenty lepiej jest zatem przewidywać w ramach bloków większych.

Dokładnie odwrotnie należy postąpić w przypadku fragmentów obrazu, które zawierają mocno złożoną treść, która podlega bardzo szybkim zmianom. Takiej treści nie da się dobrze przewidzieć za jednym zamachem, stosując jeden blok próbek o dużych rozmiarach. W tym przypadku należy działać z dużo większą skrupulatnością niż wcześniej. Szybkozmienną treść podzielić więc należy na wiele mniejszych bloków. Te mniejsze bloki znacznie ułatwią przewidywanie treści, z uwagi na zastosowanie w poszczególnych blokach dedykowanych dla nich predyktorów treści. Uzyska się w ten sposób dużo większą dokładność predykcji treści, która skompensuje z nawiązką koszt, jaki zostanie poniesiony na sygnalizację trybów użytych w blokach.

W najnowszych koderach obrazu (np. MPEG-4 AVC/H.264 czy MPEG-H HEVC/H.265), właściwe przewidywanie treści jest więc poprzedzone czynnością podziału obrazu na bloki o odpowiednim rozmiarze. Przykładowy wynik takiego podziału został przedstawiony na rysunku 4-8, który reprezentuje faktyczne decyzje, jakie dla zamieszczonego obrazu testowego zostały podjęte przez modelowe oprogramowanie HM kodera HEVC [HM]. W ramach każdego dozwolonego rozmiaru bloku koder ma do dyspozycji wcześniej określoną, dużą pulę predyktorów wartości próbek. Czyli tak naprawdę, to, co nazywamy trybem kodowania bloku, obejmuje zarówno rozmiar tego bloku jak również zastosowany w tym bloku rodzaj predyktora próbek. Pełna informacja o zastosowanych w koderze trybach (którą koniecznie trzeba przesłać do dekodera obrazu) zawiera więc instrukcję sposobu podziału obrazu na bloki oraz dane na temat rodzaju predyktora użytego w blokach. Z perspektywy kosztu przesłania tej informacji do dekodera, stopnia złożoności wyboru trybów, jak również charakterystycznych cech treści obrazów naturalnych o wysokiej rozdzielczości największe uzasadnienie znajduje dokonywanie przewidywania treści w blokach o rozmiarze od 64x64, poprzez 32x32, 16x16, 8x8, aż do 4x4. Takie między innymi rozwiązanie stało się częścią nowego kodera HEVC[35]. Pozwala ono, w połączeniu z użyciem 35 predyktorów treści bloków obrazu uzyskać ponad 2-krotną kompresję bezstratną obrazu (współczynnik kompresji na poziomie 2,1 – 2,2), w przypadku kodowania obrazów naturalnych o wysokiej rozdzielczości [Abhil17]. Jak więc widać, mechanizm adaptacyjnego wyboru rozmiaru bloków, w których przewiduje się treść, w połączeniu z liczną paletą dostępnych predyktorów próbek prowadzą do bardzo wyraźnego zwiększenia efektywności kodowania predykcyjnego obrazu.

[35] W koderze HEVC, predykcja treści bloku o wielkości 64x64 przebiega trochę nietypowo. Zamiast dokonywać predykcji całego takiego bloku, to blok ten dzieli się na 4 mniejsze bloki o rozmiarze 32x32 i przewiduje się treść tych mniejszych bloków. Jednak dla każdego z tych 4 bloków stosuje się ten sam predyktor wartości próbek, przez co tryb predykcji jest sygnalizowany tylko raz (na poziomie bloku 64x64). Proszę zauważyć, że jest to inny wariant predykcji treści niż ten, w którym w każdym z bloków 32x32 stosuje się odmienny predyktor treści (co wymaga sygnalizacji trybu cztery razy, bo na poziomie bloków 32x32).

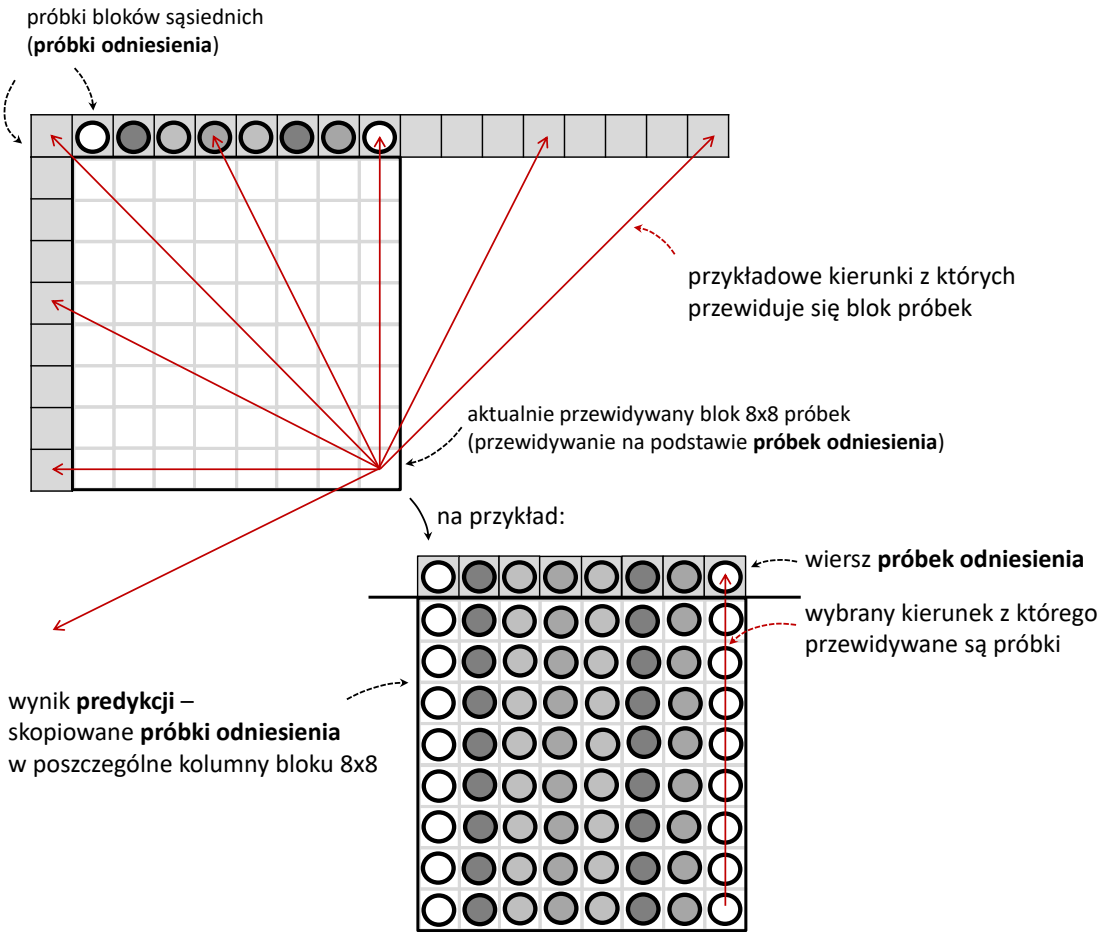
Rysunek 4-8. Obraz testowy oraz rezultat podziału tego obrazu na bloki o określonym rozmiarze. Otrzymana siatka bloków jest wynikiem podjętych przez modelowe oprogramowanie HM 10.0 kodera HEVC decyzji o sposobie kodowania treści obrazu. Przykład został opracowany z użyciem oprogramowania analizatora HEVC, którego głównym wykonawcą jest Pan Piesik Daniel.

Na koniec warto jeszcze wspomnieć o tym, w jaki dokładnie sposób można realizować przewidywanie wartości próbek bloku obrazu. W przypadku predyktora DC przewidywana wartość próbek bloku równa jest średniej wartości odpowiednich próbek bloków sąsiednich (sąsiednich w stosunku do bloku aktualnie przewidywanego). W przypadku predyktorów kątowych, wartość przewidywaną próbek można obliczać w drodze ważonej sumy np. dwóch „pewnych” próbek, do których dostęp mają koder i dekoder obrazu. Wybór tych dwóch „pewnych” próbek, w oparciu o które jest przewidywana aktualnie przetwarzana próbka, jest oczywiście podyktowany typem użytego predyktora kątowego. Z pozycji próbki, którą chcemy przewidzieć, możemy dokonać swego rodzaju rzutowania na kierunek, z którego dokonać chcemy przewidywania wartości próbki. Z linią tego kierunku sąsiadować będą „pewne” dwie próbki, które pozwolą na wydajne przewidzenie wartości bieżącej próbki. Przytoczona idea została dodatkowo zilustrowana przy pomocy rysunku 4-7 (część „b” rysunku – predyktory kątowe). W najnowszych dostępnych rozwiązaniach (np. technika HEVC) stosuje się dodatkowo **interpolację próbek** obrazu, której celem jest wyznaczenie na potrzeby predykcji nowych próbek, które znajdują się znacznie bliżej linii kierunku predykcji w stosunku do oryginalnych próbek obrazu. Takie rozwiązanie umożliwia

również w praktyce użycie szerszej niż w przypadku braku interpolacji, palety predyktorów kątowych. Większa liczba obsługiwanych kierunków predykcji poprawia skuteczność przewidywania treści.

W przypadku **bezstratnego kodowania predykcyjnego** poszczególne próbki bloków mogą być przewidywane na podstawie odpowiednich próbek bloków sąsiednich, jak również wybranych próbek tego aktualnie przewidywanego bloku, które zostały już wcześniej zakodowane i przesłane do dekodera obrazu (patrz zamieszczona na rysunku 4-7 część „b” ilustracja tego rozwiązania). We wskazanym przypadku często się stosuje takie właśnie podejście. Jednak należy przy tym pamiętać, że przewidywanie wszystkich próbek bloku realizuje ten sam, wcześniej wybrany predyktor treści.

W praktyce **kompresji stratnej** obrazu próbki przewiduje się w nieco inny sposób. Tutaj, wszystkie próbki są przewidywane na podstawie próbek, które pochodzą z sąsiednich bloków (względem bloku aktualnie przewidywanego). Czyli nie wykorzystuje się do tego celu przetworzonych już próbek bieżącego bloku. Niniejsze ograniczenie wynika z faktu późniejszego stosowania w koderze metod kompresji (jak stratne kodowanie transformatowe danych obrazu), które operują na całych blokach próbek, a nie pojedynczych próbkach. Poza tą różnicą główna idea przewidywania próbek jest taka jak w przypadku metod bezstratnego kodowania predykcyjnego.



Rysunek 4-9. Przykładowy zbiór predyktorów kierunkowych wartości próbek, wraz z ilustracją idei przewidywania próbek bloku, na podstawie próbek odniesienia, które pochodzą z innych bloków.

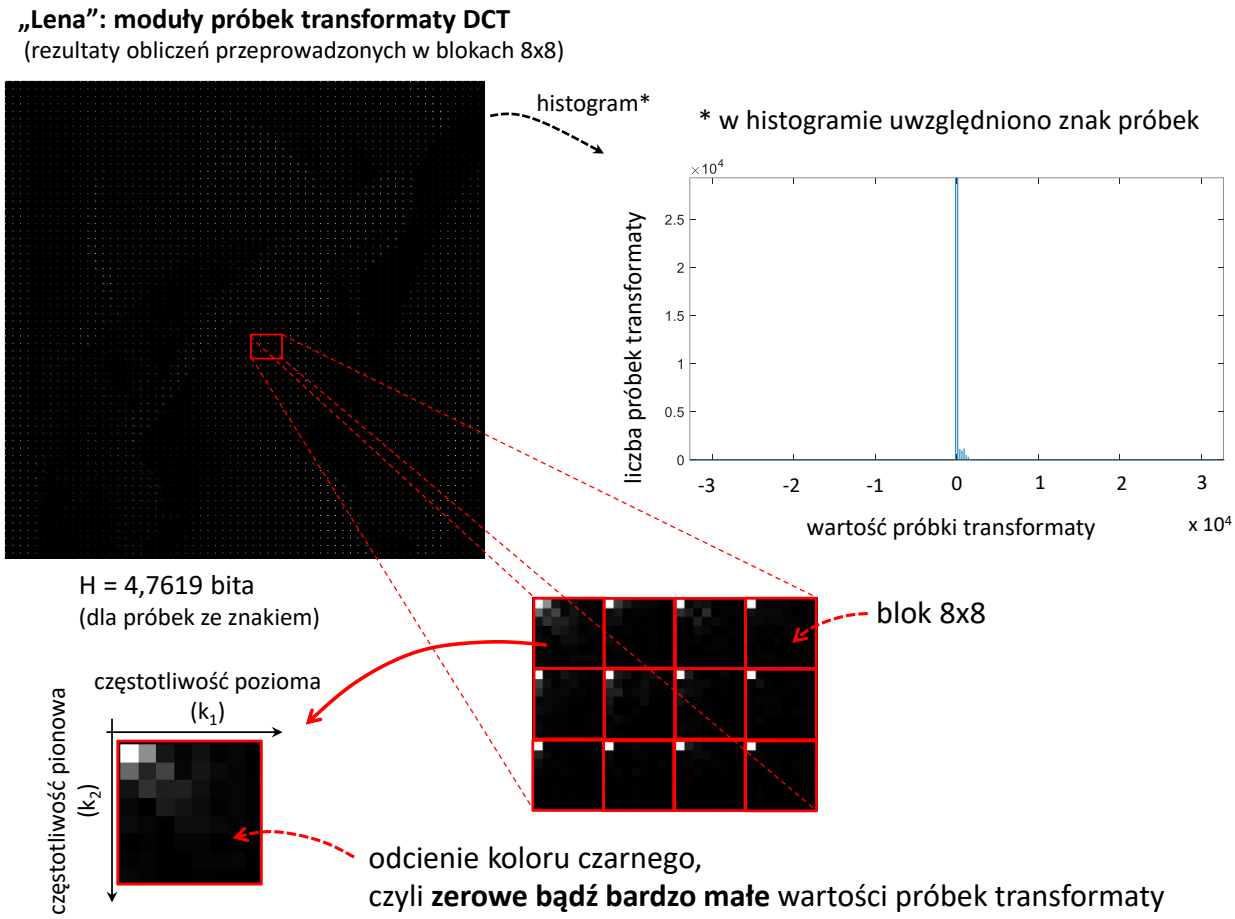
4.6. Nadmiarowość przestrzenna w obrazie – inny sposób jej redukcji

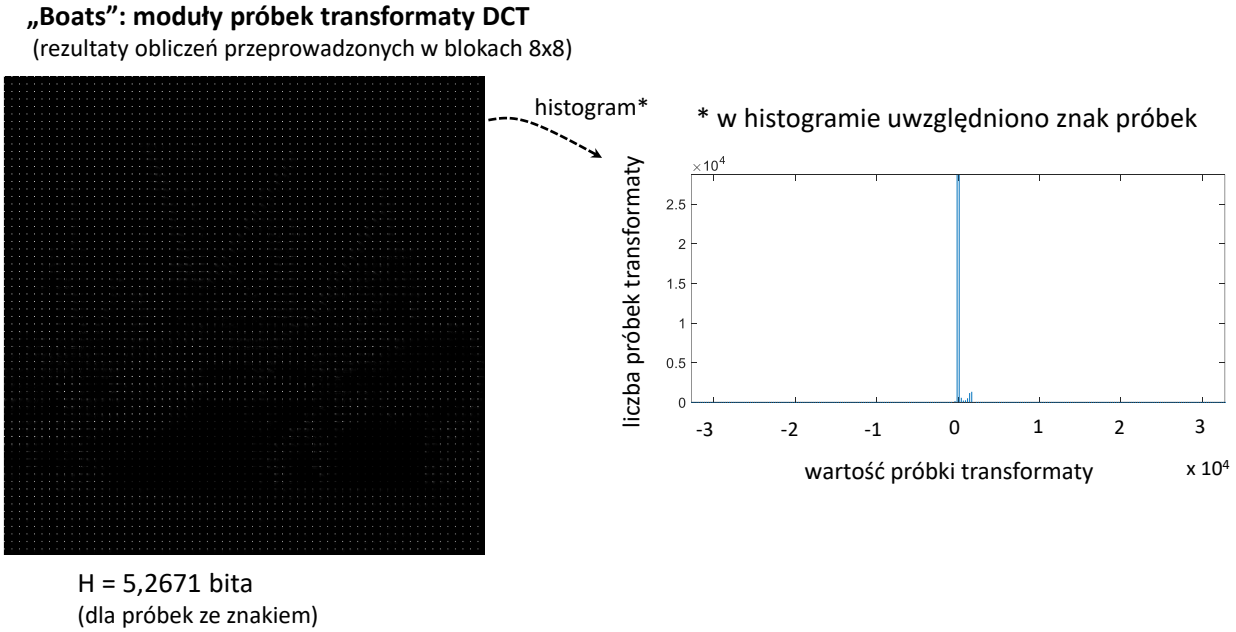
Przewidywanie próbek (czy fragmentów) obrazu nie jest oczywiście jedynym sposobem na redukcję **nadmiarowości przestrzennej** danych obrazu. Dobrym przykładem innego rozwiązania, które również pozwala ten cel osiągnąć, jest **reprezentacja obrazu w dziedzinie częstotliwości**, czyli **opis treści obrazu przy pomocy ważonej sumy składowych (funkcji) harmonicznych** o ściśle określonych częstotliwościach. W tym kontekście bardzo dobrym wyborem jest użycie **dwuwymiarowych funkcji kosinusoidalnych** (patrz Rozdział 2 – opis przekształcenia 2D-DCT) do niezależnego przedstawienia kolejnych, niewielkich fragmentów (bloków) obrazu.

Jak pokazuje praktyka reprezentacji częstotliwościowej obrazu, fragmenty obrazu o jednolitej, bądź mało złożonej treści (a tych w obrazie naturalnym jest najwięcej) daje się przedstawić w drodze sumy bardzo małej liczby niskoczęstotliwościowych kosinusów[36]. W przypadku takich fragmentów, w ich częstotliwościowej reprezentacji nie występują więc w ogóle kosinusy o wyższych częstotliwościach (czyli zerowa amplituda tych kosinusów), bądź ich amplitudy są bardzo, bardzo małe. Wyraźnie to widać na zamieszczonym dla obrazu „Lena” powiększeniu kilkunastu bloków 8x8 z próbkami 2D-DCT (patrz rysunek poniżej). Histogram amplitud składowych kosinusoidalnych (wraz z informacją o znaku tych składowych), które są obecne w obrazie (czyli histogram próbek transformaty 2D-DCT) jest więc nierównomierny, z silnym skupieniem w okolicy zerowej wartości. Z tego właśnie względu można się spodziewać, iż entropijna kompresja próbek przekształcenia kosinusowego będzie znacznie bardziej wydajna od analogicznego kodowania oryginalnych próbek obrazu.

Policzona dla próbek transformaty wartość entropii faktycznie potwierdza tę regułę. Wartości entropii próbek 2D-DCT (dla obrazów „Lena” i „Boats”) są dość zbliżone do wyników, jakie otrzymano w przypadku błędu przewidywania wartości próbek tych obrazów. Jest to $H = 4{,}7619$ bita w przypadku obrazu „Lena”, oraz $H = 5{,}2671$ bita dla obrazu „Boats”. Entropijna kompresja próbek transformaty kosinusowej umożliwiłaby więc w tym przypadku $\frac{8}{4{,}7619} = 1{,}6800$ krotną redukcję strumienia danych obrazu „Lena”. W przypadku obrazu „Boats” byłaby to redukcja $\frac{8}{5{,}2671} = 1{,}5189$ krotna. Jednak w tych przypadkach można tylko mówić o kompresji **prawie bezstratnej** (a nie całkowicie bezstratnej), ponieważ wcześniej dokonano zaokrąglenia wartości współczynników transformaty DCT do najbliższych liczb całkowitych. Niemniej jednak otrzymane wyniki z pewnością uzasadniają sensowność użycia reprezentacji częstotliwościowej danych obrazu dla celów redukcji nadmiarowości reprezentacji tych danych.

[36] Uważna lektura drugiego rozdziału pozwoli dobrze zrozumieć tę prawidłowość.





Rysunek 4-10. Moduły próbek transformaty DCT obrazów „Lena” i „Boats” oraz histogramy wartości próbek transformaty. Rezultaty obliczeń przeprowadzonych w blokach 8x8 obrazów. Jasne punkty reprezentują próbki transformaty o dużych wartościach. Dla przedstawionych obrazów entropia próbek transformaty wynosi $H = 4{,}7619$ bita dla obrazu „Lena” i $H = 5{,}2671$ bita w przypadku obrazu „Boats”.

4.7. Jeszcze lepsze rozwiązanie, czyli łączne użycie metody przewidywania wartości próbek z opisem wyniku funkcjami kosinusoidalnymi

W stosunku do rezultatów dwóch wcześniejszych metod, czyli metody przewidywania wartości próbek obrazu, czy metody bezpośredniego opisu próbek przy pomocy funkcji kosinusoidalnych, nadmiarowość przestrzenną próbek obrazu można jeszcze silniej zredukować, jeśli obie te metody zastosuje się w sposób łączny. Proste wyliczenia pokazują, że jeśli najpierw dokona się przewidywania wartości próbek, a następnie wynikowe wartości (czyli błąd predykcji próbek obrazu) opisze się funkcjami kosinusoidalnymi, to otrzymane w taki sposób amplitudy kosinusów (wraz z informacją o znaku) będą miały jeszcze mniejszą entropię niż entropia próbek błędu predykcji, czy amplitud kosinusów (wraz z informacją o znaku) wyznaczonych dla oryginalnych próbek obrazu. Można to łatwo zaobserwować, porównując zamieszczone na rysunkach 4-11 i 4-12 wyniki wartości entropii poszczególnych typów danych.

W związku z powyższym współczesne kodery obrazu faktycznie stosują wspomniane rozwiązanie. Czyli najpierw przewidują treść obrazu, a następnie wynik tego przewidywania (czyli sygnał błędu predykcji próbek) reprezentują odpowiednim zbiorem kosinusów. Na pierwszy rzut oka takie podejście może się wydawać nawet dziwne, niezrozumiałe. W końcu starsze kodery obrazu (np. JPEG, czy wewnątrzobrazowy tryb kodowania MPEG-2) przyzwyczaiły nas do tego, żeby kosinusami opisywać samą treść obrazu, a nie wynik jej przewidywania. Jednak już po chwili zastanowienia łatwo można zrozumieć, dlaczego warto jest opisywać kosinusami właśnie błąd przewidywania próbek. Przekształcenie kosinusowe (2D-DCT) pozwala w sposób bardzo zwięzły, zwarty (czyli przy pomocy małej liczby kosinusów) opisywać dane, które wykazują silne podobieństwo, co było już wielokrotnie podkreślane w tej książce. Taką cechę wykazują wartości sąsiednich próbek obrazu naturalnego, ale w jeszcze większym stopniu również próbki błędu przewidywania tych wartości. Dobrze działający predyktor wartości próbek popełnia zwykle bardzo małą lub wręcz zerową pomyłkę przewidując treść. Tak jest z pewnością w przypadku użycia zaawansowanego predyktora, czyli takiego, który dostosowuje swoje działanie do lokalnych cech treści obrazu. Postać sygnału błędu predykcji treści jest więc bardzo, bardzo prosta. Przez to, jeszcze bardziej predysponuje go do tego, żeby opisywać go przy pomocy niewielkiego zbioru kosinusów, w porównaniu z sygnałem, który bezpośrednio reprezentuje treść obrazu.

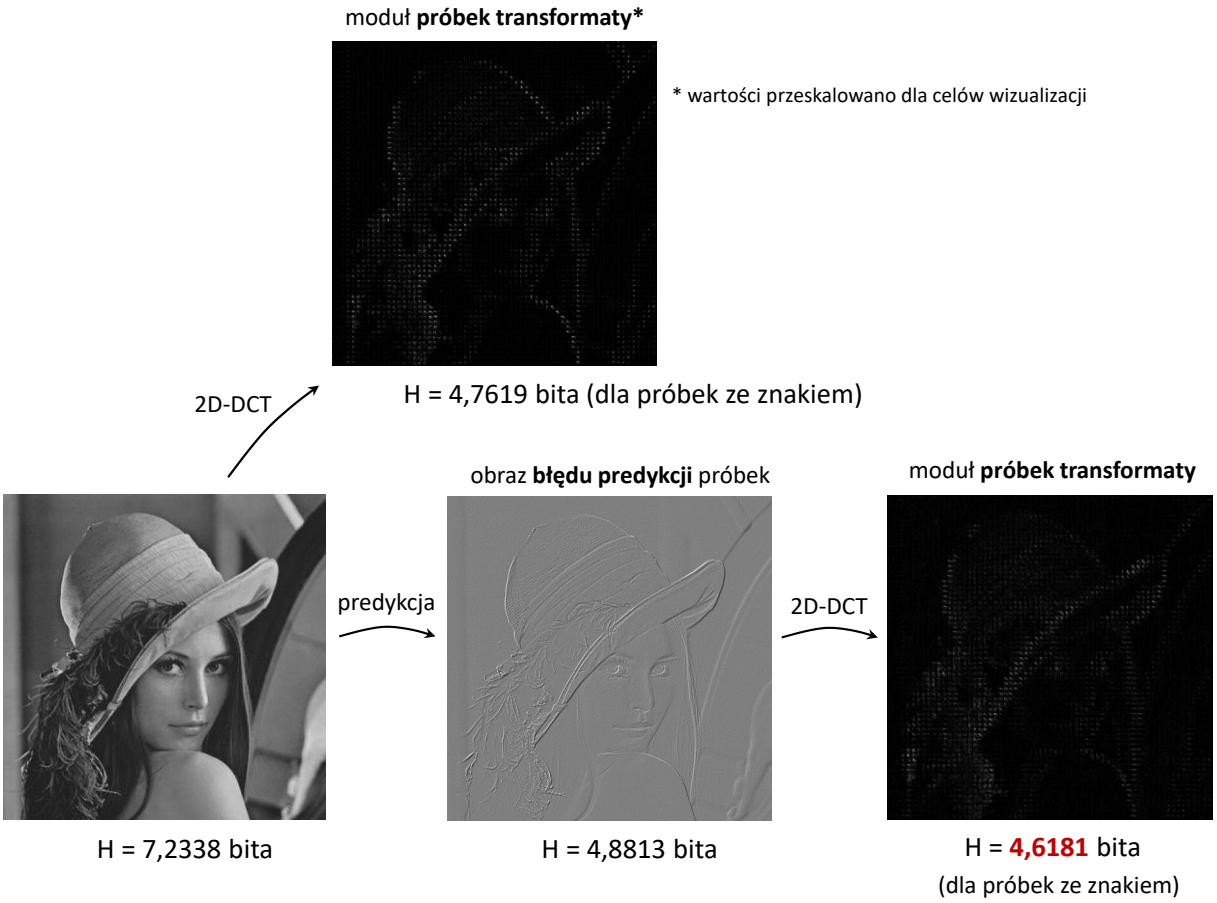
W świetle przedstawionych wyników może się wydawać, że połączenie obu technik (przewidywania treści obrazu z późniejszym opisem wyniku funkcjami harmonicznymi, czyli kosinusami) daje ciągle nieduże możliwości kompresji danych obrazu. W rozważanych przypadkach była to $\frac{8}{4,6181} = 1,7323$ oraz $\frac{8}{5,0857} = 1,5730$ – krotna redukcja objętości danych, opisujących obrazy „Lena” i „Boats”. Należy uwzględnić jednak dwie rzeczy. Po pierwsze, przykłady dotyczą wyników kodowania (prawie) bezstratnego. Dzięki lekturze kolejnych punktów książki będzie można się przekonać, że reprezentacja danych obrazu funkcjami kosinusoidalnymi otwiera perspektywę bardzo wydajnej, stratnej kompresji danych obrazu. Po drugie, sposób, w jaki w przedstawionych przykładach zostały zrealizowane obie wspomniane techniki był trywialny. Samo przewidywanie próbek obrazu przebiegało za każdym razem w ten sam sposób – była to zawsze średnia z wartości trzech próbek, które sąsiadują z aktualnie przewidywaną próbką. Sposób predykcji próbek był więc nieadaptacyjny, czyli nie pozwalał na wybór najlepszego sposobu przewidywania wartości próbek dla bieżącego charakteru treści fragmentu obrazu. W przypadku drugiej techniki funkcjami kosinusoidalnymi opisywano bloki wartości błędu przewidywania próbek, których rozmiar była taki sam, wcześniej z góry ustalony. Były to bloki o stałym rozmiarze 8x8 wartości. Tak więc, sam sposób realizacji w koderze obu technik ma również bardzo duże

znaczenie. Badania ostatnich kilkunastu lat pokazują, że znacznie lepsze wyniki kompresji danych obrazu gwarantują rozwiązania, w których:

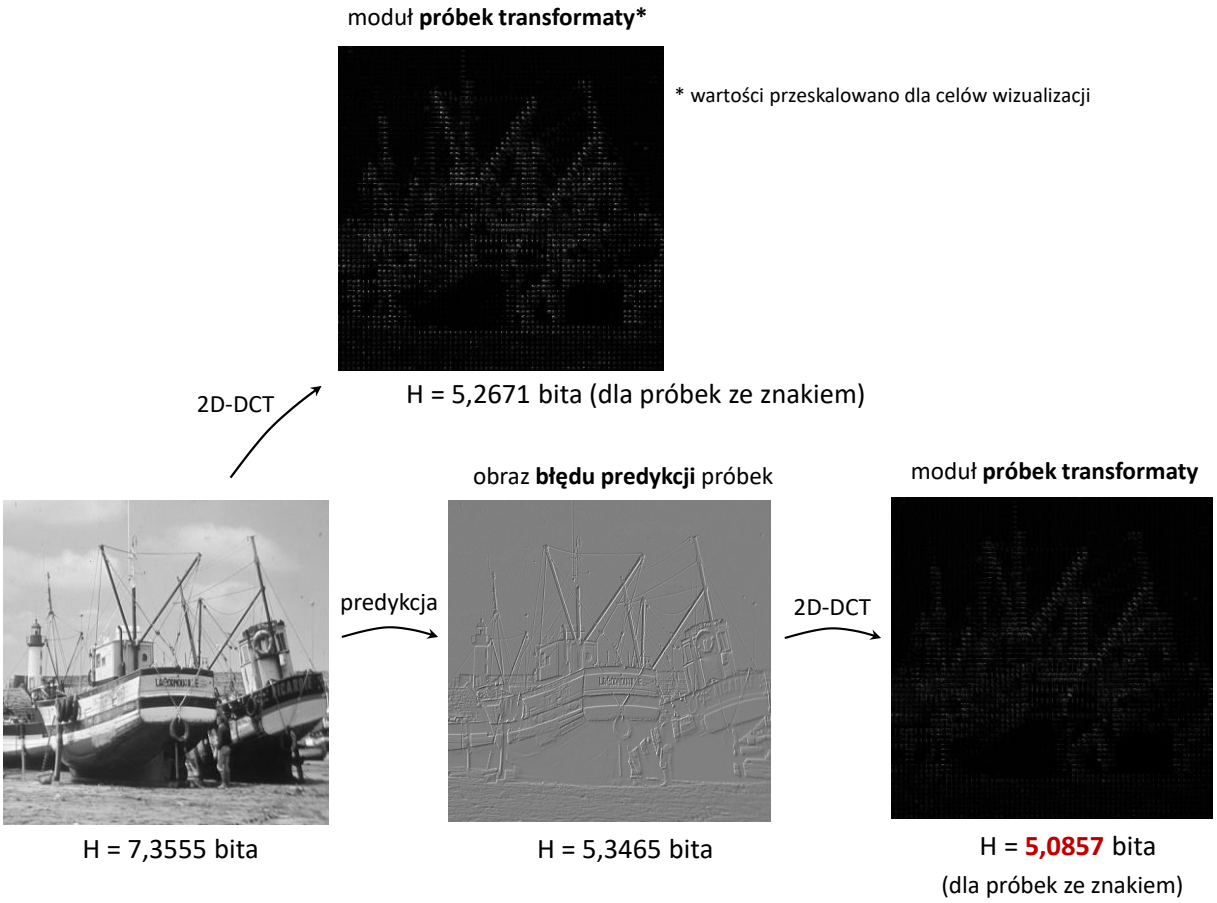
1. Przewidywanie treści obrazu jest realizowane w blokach o zmiennym rozmiarze. W przypadku najnowszych koderów obrazu (np. koder HEVC) są to bloki o rozmiarze od 4x4, poprzez 8x8, aż do 64x64). Dodatkowo, treść każdego z bloków może być przewidywana na wiele różnych sposobów (kilkadziesiąt sposobów predykcji treści bloku w najnowszych koderach obrazu). W takim wypadku, szeroki wachlarz możliwych rozmiarów bloków, oraz duża liczba dostępnych sposobów predykcji treści tych bloków dają koderowi możliwość wyboru najlepszego wariantu przewidywania treści, który dla danego fragmentu obrazu daje faktycznie najlepsze rezultaty predykcji (patrz punkt 4.5).

2. Przy pomocy kosinusów opisuje się bloki próbek błędu predykcji treści, jednak dla poszczególnych fragmentów obrazu rozmiar tych bloków pozostawia się w gestii decyzji kodera. W najnowszych rozwiązaniach (np. HEVC) korzysta się z bloków transformaty o rozmiarach 4x4, 8x8, 16x16 i 32x32. Analizując zatem charakter sygnału błędu przewidywania próbek w poszczególnych fragmentach obrazu koder dzieli obraz ściśle określoną siatką bloków o zmiennym rozmiarze, która zapewnia wysoką wydajność reprezentacji danych przy pomocy funkcji kosinusoidalnych.

Należy w tym miejscu mocno podkreślić, iż ustalone przez koder siatki bloków dla obu technik (przewidywania wartości próbek oraz reprezentacji błędu funkcjami kosinusowymi) nie muszą być takie same, czyli mogą się między sobą różnić.



Rysunek 4-11. Porównanie wartości entropii następujących danych obrazu „Lena”: 1) oryginalne próbki obrazu, 2) błąd predykcji wartości próbek, 3) współczynniki transformaty kosinusowej dla próbek obrazu, 4) współczynniki transformaty kosinusowej dla błędu predykcji próbek.



Rysunek 4-12. Porównanie wartości entropii następujących danych obrazu „Boats”: 1) oryginalne próbki obrazu, 2) błąd predykcji wartości próbek, 3) współczynniki transformaty kosinusowej dla próbek obrazu, 4) współczynniki transformaty kosinusowej dla błędu predykcji próbek.

4.8. Usunięcie części informacji o obrazie jako sposób na istotne zwiększenie stopnia kompresji danych

Przewidywanie wartości próbek obrazu, czy opisywanie próbek „kolekcją” składowych harmonicznych dają możliwość dość wyraźnej, bo kilkukrotnej (w praktyce 1,5-3 krotnej) kompresji danych obrazowych[37]. Jednak z punktu widzenia wielu zastosowań, jak fotografia (przynajmniej niektóre jej obszary), czy transmisja obrazów w sieciach teleinformatycznych taki stopień kompresji danych jest ciągle niewystarczający. Z perspektywy tych zastosowań, istnieje potrzeba znacznie silniejszej redukcji zbioru danych, który opisuje obraz, np. zmniejszenia objętości danych kilkadziesiąt razy.

Doświadczenie pokazuje jednak, że tak silnej kompresji nie da się osiągnąć samymi tylko technikami **kodowania bezstratnego** obrazów, czyli kodowania, w którym bezbłędnie koduje się wszystkie dane obrazu (przez co, obraz oryginalny jest identyczny z tym, który jest później odtwarzany w dekoderze obrazu). W tym przypadku należy zrezygnować z takiego podejścia na rzecz **kodowania stratnego**, w którym część informacji o obrazie całkowicie się pomija (czyli usuwa na etapie kompresji), bądź koduje z mniejszą dokładnością. Wykluczenie części informacji

[37] Taki wynik zostałby otrzymany gdyby połączyć wspomniane metody z techniką entropijnego kodowania odpowiednich danych.

o treści obrazu bądź jej zakodowanie z mniejszą dokładnością, prowadzą do ogromnych oszczędności bitowych reprezentacji danych obrazu. Jednak ceną tej oszczędności jest bezpowrotna (nieodwracalna) utrata części wiedzy o treści obrazu, co skutkuje pogorszeniem jakości kodowanego obrazu. Pogorszeniem tym większym, im większa część danych o obrazie zostanie całkowicie usunięta, bądź mniej dokładnie zakodowana.

W związku z powyższym faktem, osiągnięcie najlepszego kompromisu pomiędzy stopniem kompresji danych obrazu i jakością zakodowanego obrazu jest w przypadku kompresji stratnej prawdziwą sztuką. Kluczowe jest tutaj oczywiście następujące pytanie: które dane obrazu najlepiej jest usuwać bądź zakodować z mniejszą dokładnością, żeby doprowadzić do istotnej redukcji strumienia zakodowanych danych, przy jednoczesnym utrzymaniu wrażenia wysokiej jakości obrazu zakodowanego?

Odpowiedzi na to pytanie dostarcza nam wiedza na temat możliwości percepcji treści obrazu przez człowieka. Z punktu widzenia kompresji danych obrazowych szczególnie istotne są ograniczenia w zakresie zdolności odbioru informacji o obrazie, która opisuje drobne detale jego treści. Podczas gdy z dużą dokładnością potrafimy „rejestrować” niskoczęstotliwościowe elementy treści, czyli takie, które powoli się zmieniają w obrazie (odpowiada im prosta tekstura), to możliwości postrzegania danych wysokoczęstotliwościowych (czyli opisujących złożone tekstury, które są obfite w krawędzie i inne drobne elementy treści) są już mocno ograniczone. Zamiast więc z jednakową dokładnością kodować wszystkie dane obrazu (nisko- oraz wysokoczęstotliwościowe), lepiej jest dokładniej reprezentować informację niskoczęstotliwościową, a mniej dokładnie wysokoczęstotliwościową. Przykłady tych dwóch typów danych obrazu zostały przedstawione na poniższym rysunku. Takie podejście ma głęboki sens, ponieważ wprowadza zniekształcenia kompresji stratnej do tych fragmentów obrazu, w których trudniej jest je człowiekowi dostrzec. Podejście to pozwala więc uzyskać bardzo dobrą relację bitowego kosztu reprezentacji obrazu i jakości zakodowanego obrazu.

Rysunek 4-13. Przykładowe fragmenty obrazu, które znacząco różnią się istotnością z punktu widzenia percepcji treści obrazu przez człowieka. Ilustracja fragmentów, które zawierają nisko- oraz wysokoczęstotliwościowe dane obrazu. Pierwszy typ danych należy zakodować dokładniej niż typ drugi.

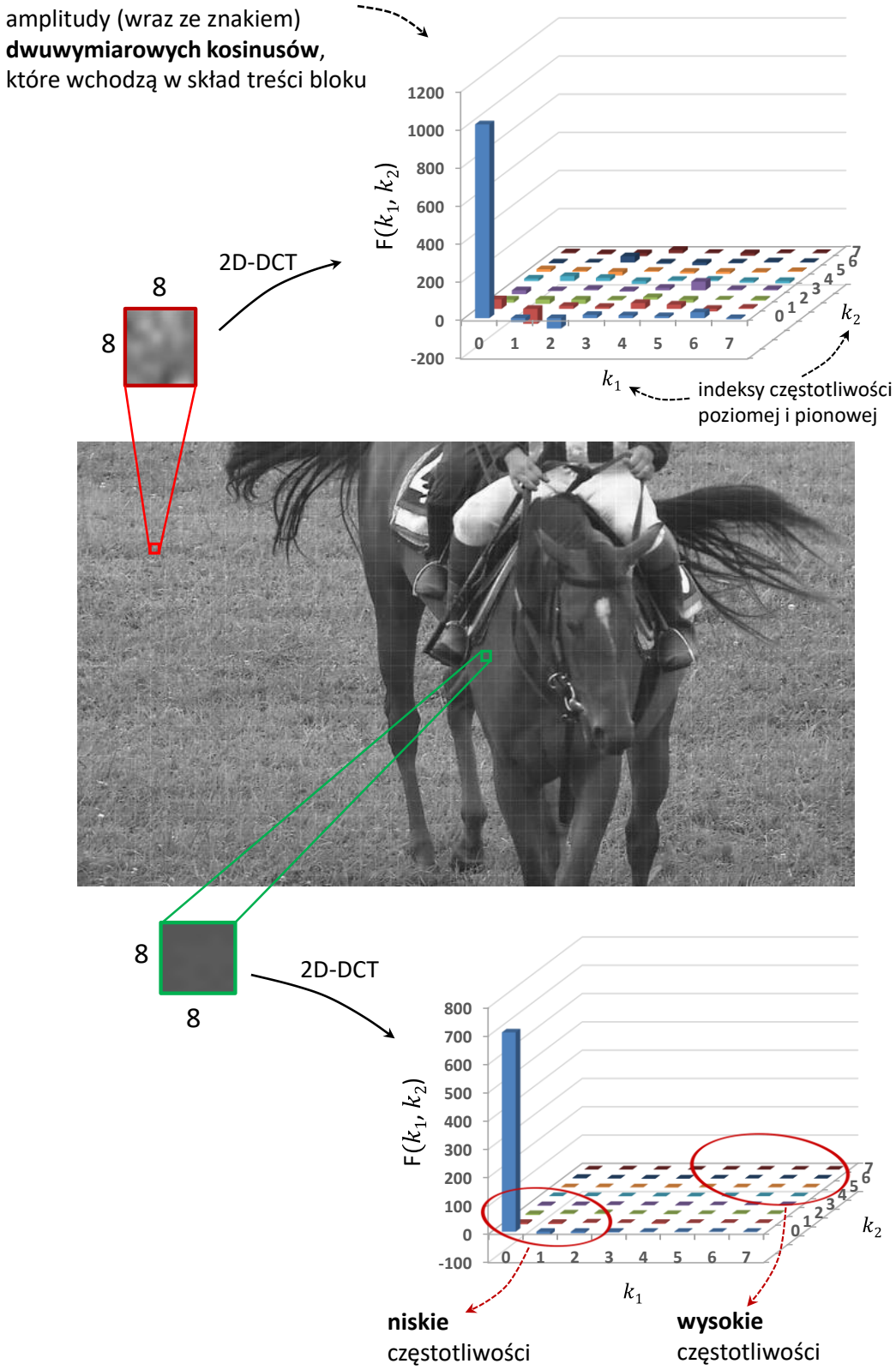
Reguła niejednakowej dokładności, z jaką opisuje się nisko- i wysokoczęstotliwościowe dane obrazu jest podstawą działania współczesnych koderów stratnych obrazu. To, jak to jest robione w szczegółach, jest tematem kolejnych punktów książki.

4.9. Kompresja obrazu w dziedzinie częstotliwości – furtka do wydajnej realizacji idei kodowania stratnego

Realizacja idei dokładniejszej reprezentacji wolnozmiennych (czyli niskoczęstotliwościowych) elementów treści obrazu, z jednoczesnym, mniej dokładnym kodowaniem składowych, które odpowiadają za opis szybkich zmian treści (czyli składniki wysokoczęstotliwościowe) wymaga właściwej ekstrakcji tych składników z treści obrazu. Potrzebna jest zatem dokładna wiedza o tym, jaki jest charakter zmian treści obrazu (wolnozmienny, czy szybkozmienny), jaka jest częstotliwość (prędkość) tych zmian, oraz w których dokładnie fragmentach obrazu wspomniane zmiany treści występują. Takiej wiedzy dostarcza nam analiza „zawartości” częstotliwościowej obrazu, przeprowadzana niezależnie w poszczególnych jego fragmentach (blokach).

Wydajnym narzędziem do przeprowadzenia takiej analizy jest dwuwymiarowe dyskretne przekształcenie kosinusowe (2D-DCT). Można się o tym przekonać studiując zamieszczony w rozdziale 2 opis tego przekształcenia. Ocenia ono „zawartość” częstotliwościową fragmentu obrazu, „rozkładając” jego treść na dwuwymiarowe funkcje kosinusoidalne o ściśle określonej

częstotliwości, amplitudzie oraz znaku (dodatni lub ujemny). Jest to robione w drodze odpowiednich przekształceń matematycznych (patrz Rozdział 2), których działanie sprowadza się do znalezienia odpowiedzi na następujące pytanie: suma jakich dwuwymiarowych kosinusów pozwala w pełni odtworzyć wartości próbek badanego fragmentu obrazu? Rezultatem przekształcenia jest więc nie tylko ogólnikowa wiedza o tym, czy treść fragmentu jest złożona czy prosta, czy zmiany treści mają charakter powolny czy szybki, ale wynikiem jest również dokładna informacja o „ilościowym” udziale poszczególnych „częstotliwości” w treści analizowanego fragmentu obrazu. Można się o tym przekonać analizując zamieszczony na rysunku 4-14 wynik analizy częstotliwościowej treści dwóch wybranych bloków obrazu.



Rysunek 4-14. „Zawartość” częstotliwościowa dwóch wybranych bloków obrazu. Wykresy słupkowe przedstawiają amplitudy (wraz ze znakiem) dwuwymiarowych funkcji kosinusoidalnych (o ściśle określonej wartości częstotliwości poziomej i pionowej), które wchodzą w skład treści badanych bloków.

Tak szczegółowa wiedza o „zawartości” treści poszczególnych fragmentów obrazu otwiera furtkę do wydajnej realizacji idei kompresji stratnej obrazu. Widząc, jaki jest „ilościowy” udział poszczególnych kosinusów (kosinusów o danej częstotliwości) w przedstawieniu treści fragmentu obrazu możemy na przykład łatwo wskazać składowe (kosinusy) o najmniejszej amplitudzie. Jeśli ta niewielka amplituda dotyczy wysokoczęstotliwościowych składowych, to z punktu widzenia percepcji treści obrazu, ich obecność w obrazie ma bardzo ograniczone znaczenie. Dlatego, można się nawet pokusić o całkowite ich pominięcie w trakcie kodowania danych. Istnieje również możliwość mniej radykalnego działania, polegającego na tym, że koduje się co prawda poszczególne składowe, ale redukuje się dokładność ich reprezentacji w koderze. W takim przypadku, najlepszym rozwiązaniem jest niezależne ustalenie dokładności reprezentacji składowej o danej wartości częstotliwości, co wynika wprost z przesłanek percepcji poszczególnych „częstotliwości” przez człowieka. Zgodnie z tymi przesłankami, kosinusy o niskich częstotliwościach należy kodować z większą dokładnością (bo w przypadku tych składowych człowiek wykazuje większą zdolność ich postrzegania), a mniejszą dokładność reprezentacji można zastosować dla kosinusów wysokoczęstotliwościowych. To właśnie możliwość „indywidualnego” traktowania poszczególnych składowych, które reprezentują w obrazie zgoła inne elementy treści (elementy wolnozmienne i szybkozmienne), jest przyczynkiem bardzo wydajnej kompresji danych obrazu. Takiej możliwości, przynajmniej w tak dużym stopniu, trudno jest szukać, gdyby realizować kompresję obrazu bezpośrednio w dziedzinie jego próbek. Należy tutaj raz jeszcze mocno podkreślić, że usunięcie części kosinusów, czy zmniejszenie dokładności ich reprezentacji, są w efekcie źródłem bardzo znaczących oszczędności bitowych reprezentacji danych. Jednak skutkiem ubocznym tych operacji jest bezpowrotna utrata części informacji o kodowanym obrazie, co skutkuje pogorszeniem jego jakości. Jednak właściwy dobór dokładności reprezentacji poszczególnych składowych (właśnie z uwzględnieniem zdolności postrzegania składowych o danej częstotliwości) pozwala osiągnąć bardzo korzystny kompromis pomiędzy stopniem kompresji danych a jakością zakodowanego obrazu.

W koderach obrazu dokładność reprezentacji poszczególnych kosinusów jest ustalana poprzez odpowiednie **kwantowanie** ich amplitud. Kwantowanie danej składowej realizuje się poprzez podzielenie jej amplitudy przez pewną liczbę większą od 1, oznaczoną np. przez Q (np. przez Q=16), a następnie zaokrąglenie otrzymanego w ten sposób wyniku do (najbliższej) liczby całkowitej. Nietrudno się domyślić, iż w efekcie tych operacji wiele różnych wartości wejściowych (Q różnych wartości) da w rezultacie ten sam wynik kwantowania amplitud. Kwantowanie redukuje zatem zbiór możliwych wartości, jakie będą przyjmować nowe, skwantowane amplitudy[38]. Robi to tym silniej, im większa jest wartość Q, przez którą dzielona jest wartość amplitudy kosinusa. W takiej sytuacji, nie zawsze jest więc możliwe bezbłędne odtworzenie w dekoderze obrazu oryginalnej wartości amplitudy, na podstawie jej wartości skwantowanej. Tak więc, część informacji o oryginalnej wartości amplitudy jest nieodwracalnie tracona, co jest właśnie powodem spadku jakości kodowanego obrazu. Jednak jak już wielokrotnie podkreślano, korzyścią jest znaczące ograniczenie rozmiaru danych, które będą opisywać zakodowany obraz.

W przedstawionej idei stratnej kompresji obrazu koduje się dane, które są wynikiem wcześniejszej transformacji bloku próbek do nowej dziedziny, dziedziny częstotliwości. Dlatego taki sposób kodowania danych obrazu jest powszechnie nazywany **kodowaniem transformatowym** [Skarb98, Doma10]. W przypadku tego kodowania istnieje możliwość znaczącej redukcji strumienia danych obrazu, nie powodując przy tym dużego spadku jakości obrazu. Dodatkowo, zastosowanie odpowiednio silnego kwantowania amplitud składowych kosinusoidalnych, pozwala w płynny sposób regulować rozmiar oraz jakość zakodowanego obrazu.

[38] Przedstawiony sposób kwantowania wartości daje więc takie same wyniki co przedstawiona w punkcie 1.3.1 „schodkowa” funkcja kwantyzatora równomiernego (patrz rysunek 1-6). Wartość Q jest tutaj niczym innym jak szerokością kroku kwantyzacji.

Przedstawiona powyżej idea stratnej kompresji obrazu zdecydowanie dominuje w praktycznych zastosowaniach i jest w literaturze znana jako **kodowanie transformatowe** obrazu.

4.10. Praktyczna realizacja techniki kodowania transformatowego

Jak dotąd przedstawiona została tylko główna, widziana z lotu ptaka, idea metody kodowania transformatowego danych obrazu. Jednak z punktu widzenia najważniejszych parametrów tej metody, czyli efektywności kompresji obrazu oraz złożoności obliczeniowej kodowania i dekodowania danych, niezwykle kluczowe są w tej idei następujące kwestie:

1. Jak, w szczegółach, realizowany jest opis treści obrazu w dziedzinie częstotliwości? Z lektury drugiego rozdziału wiadomo, że opisu tego najlepiej jest dokonywać niezależnie dla kolejnych fragmentów (bloków) obrazu. W praktyce więc, powyższe pytanie dotyczy przyjętego w koderze sposobu podziału obrazu na bloki, których treść jest następnie reprezentowana zbiorem kosinusów o określonych częstotliwościach. No i podstawowa kwestia, czy jest to sztywny podział na bloki o jednakowym rozmiarze (np. 8x8 próbek obrazu), czy może koder ma możliwość autonomicznego ustalenia najlepszych rozmiarów dla poszczególnych bloków, posiłkując się wiedzą o charakterze treści, która jest w tych blokach zawarta?

2. Jak przebiega kwantyzacja amplitud poszczególnych kosinusów oraz jaki jest algorytm kodowania tych amplitud już po ich skwantowaniu?

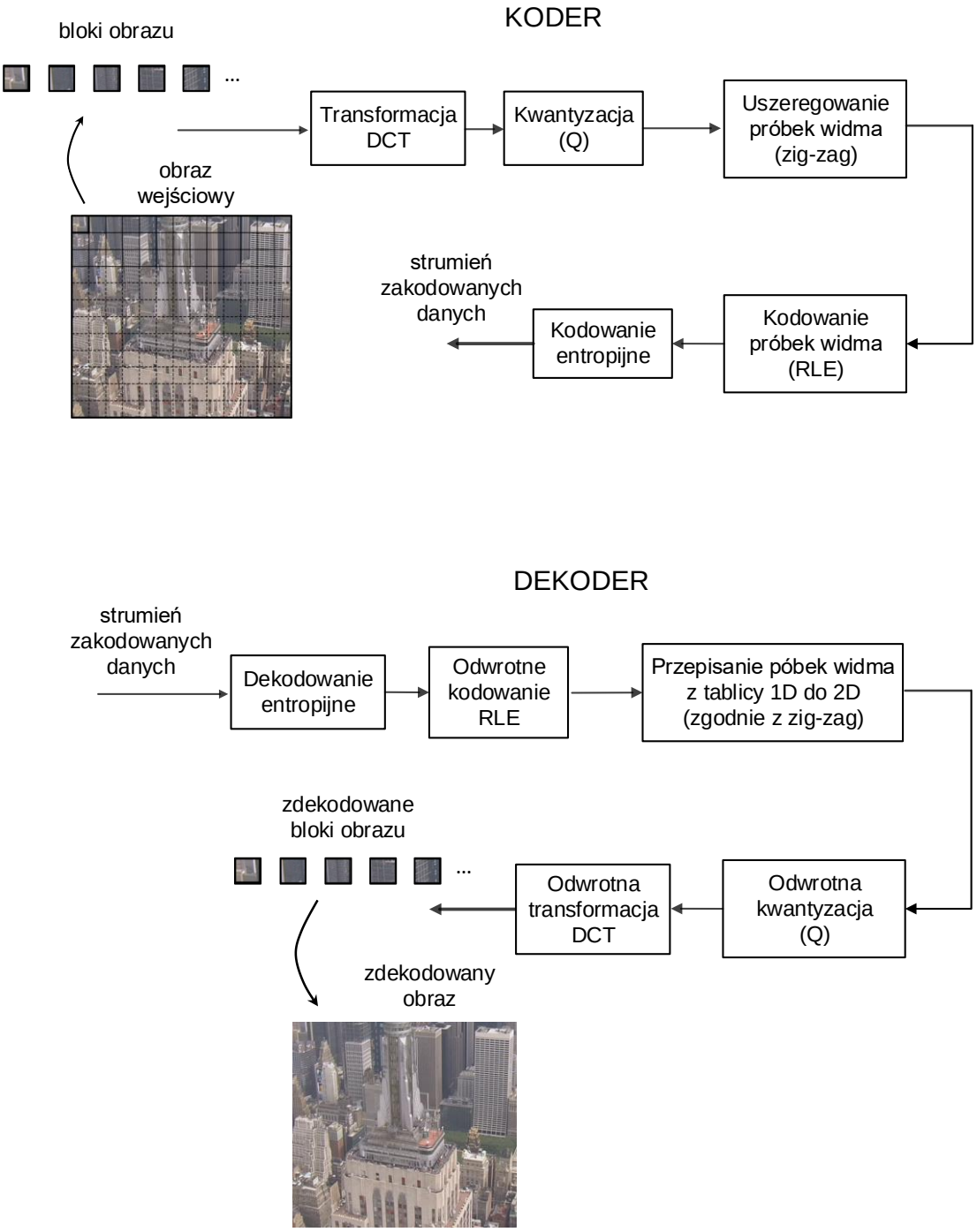
Doświadczenie pokazuje, że dwa różne kodery obrazu, każdy realizujący ideę kompresji transformatowej, może z punktu widzenia osiąganych wyników wydajności kodowania danych dzielić wprost przepaść. To, w jaki konkretnie sposób koder obrazu rozwiązuje powyższe kwestie, ma więc znaczenie fundamentalne. Żeby się o tym przekonać, w kolejnych dwóch punktach przedstawione zostały dwie, istotnie się między sobą różniące realizacje idei kodowania transformatowego obrazu – technika prosta oraz bardzo zaawansowana.

4.10.1. Kodowanie transformatowe w bardzo prostym wydaniu

Dobrymi przykładami prostej realizacji idei kompresji transformatowej są techniki kodowania, które spotkać można w starszych koderach obrazu, jak np. w koderze JPEG [JPEG], czy w koderze MPEG-2 (tylko kodowanie wewnątrzobrazowe) [MPEG-2]. Przyjęte w obu koderach rozwiązania są bliźniaczo podobne (oczywiście w odniesieniu do kompresji statycznego, czyli nieruchomego obrazu), mało skomplikowane i stanowią osiągnięcie pierwszej połowy lat 90-tych ubiegłego stulecia. Z technicznego, inżynierskiego punktu widzenia, są to więc rozwiązania bardzo stare.

Ogólny schemat blokowy tamtych rozwiązań (w odniesieniu do kodera i dekodera obrazu) został przedstawiony na rysunku 4-15. Zgodnie z tym schematem, przywołane kodery obrazu realizują ciąg prostych obliczeniowo operacji, co jak zostanie dokładnie pokazane ma swoje odzwierciedlenie w umiarkowanej, żeby nie powiedzieć niskiej, efektywności kompresji danych obrazu. Jednak ten niewielki stopień zaawansowania przyjętych wówczas algorytmów wynikał w bardzo dużym stopniu z niewielkiej, jak na dzisiejsze uwarunkowania, mocy obliczeniowej dostępnych w tamtym czasie jednostek obliczeniowych (procesory, układy programowalne). Jak

można bardzo łatwo zauważyć dekoder wykonuje analogiczne czynności, co koder, tylko w odwrotnej kolejności.



Rysunek 4-15. Schemat blokowy **prostego** kodeka transformatowego obrazu (koder + dekoder). Schemat ten odzwierciedla sposób kompresji obrazów w starych już koderach JPEG oraz MPEG-2 (tylko tryb wewnątrzobrazowy).

4.10.1.1. Podział obrazu na małe fragmenty (bloki)

Idei kompresji transformatowej nie można realizować na całym obrazie w jednym kroku. Takie podejście byłoby skrajnie niewydajne. **Po pierwsze**, wyznaczenie zbioru kosinusów, który

za jednym zamachem opisywałby treść całego obrazu wiązałoby się z ogromnymi nakładami obliczeniowymi. Pochłaniałoby również bardzo duże zasoby pamięci w koderze i dekoderze obrazu. Z punktu widzenia praktycznej realizacji takie rozwiązanie rodziłoby więc olbrzymie problemy. **Po drugie**, jakichkolwiek zalet trudno również szukać w samych rezultatach tak realizowanego przekształcenia. Wyliczony dla treści całego obrazu zbiór kosinusów byłby bardzo złożony. Zawierałby on funkcje kosinusoidalne o częstotliwościach niskich, średnich i wysokich, przez co bitowa reprezentacja tego bogatego zbioru składowych byłaby bardzo kosztowna.

Osiągnięcie wysokiej efektywności kodowania danych obrazu jest więc wyłącznie możliwe wtedy, kiedy kodowanie odbywa się w ramach małych zbiorów danych, czyli jest realizowane w niewielkich blokach obrazu. W tym przypadku, próbki tych niedużych bloków będą wykazywać silne podobieństwo (wartości sąsiednich próbek będą do siebie bardzo podobne), co jak już wykazano w rozdziale 2 jest koniecznym warunkiem do tego, żeby treść bloków dało się opisywać przy pomocy małego (lub nawet bardzo małego) zbioru kosinusów. Oczywiście, im mniejszy jest ten zbiór, tym mniejszy jest koszt bitowy transmisji kosinusów do dekodera obrazu. Operując na małych blokach próbek znacząco mniejszy będzie również pamięciowy i obliczeniowy koszt wyliczenia zbioru kosinusów, które będą opisywać treść bloków, co ma bardzo duże znaczenie z perspektywy praktycznej realizacji algorytmu kompresji i dekompresji.

W związku z powyższymi przesłankami właściwy opis treści obrazu przy pomocy funkcji kosinusoidalnych jest poprzedzony podziałem obrazu na małe fragmenty. W przypadku prostej realizacji idei kompresji transformatowej obraz jest dzielony na bloki o jednakowym, z góry ustalonym rozmiarze. W przypadku koderów JPEG oraz MPEG-2 jest to podział obrazu na bloki o rozmiarze 8x8 próbek. Rezultat takiego podziału obrazu na siatkę bloków został przedstawiony na rysunku 4-16.

Rysunek 4-16. Obraz o rozdzielczości przestrzennej 832x480 z nałożoną siatką bloków o rozmiarze 8x8 próbek. Niezależnie, w kolejnych blokach 8x8 jest realizowana idea kompresji transformatowej obrazu.

4.10.1.2. Reprezentacja treści bloków w dziedzinie częstotliwości

Treść obrazu, która jest zawarta w pojedynczym, małym bloku 8x8 obrazu jest zwykle bardzo prosta. Kolejne próbki takiego niedużego bloku wykazują bardzo silne podobieństwo. Taki charakter sygnału otwiera możliwość bardzo wydajnej jego reprezentacji, jeśli opisze się go **dwuwymiarowymi funkcjami kosinusoidalnymi**. Ta kwestia była już przedmiotem

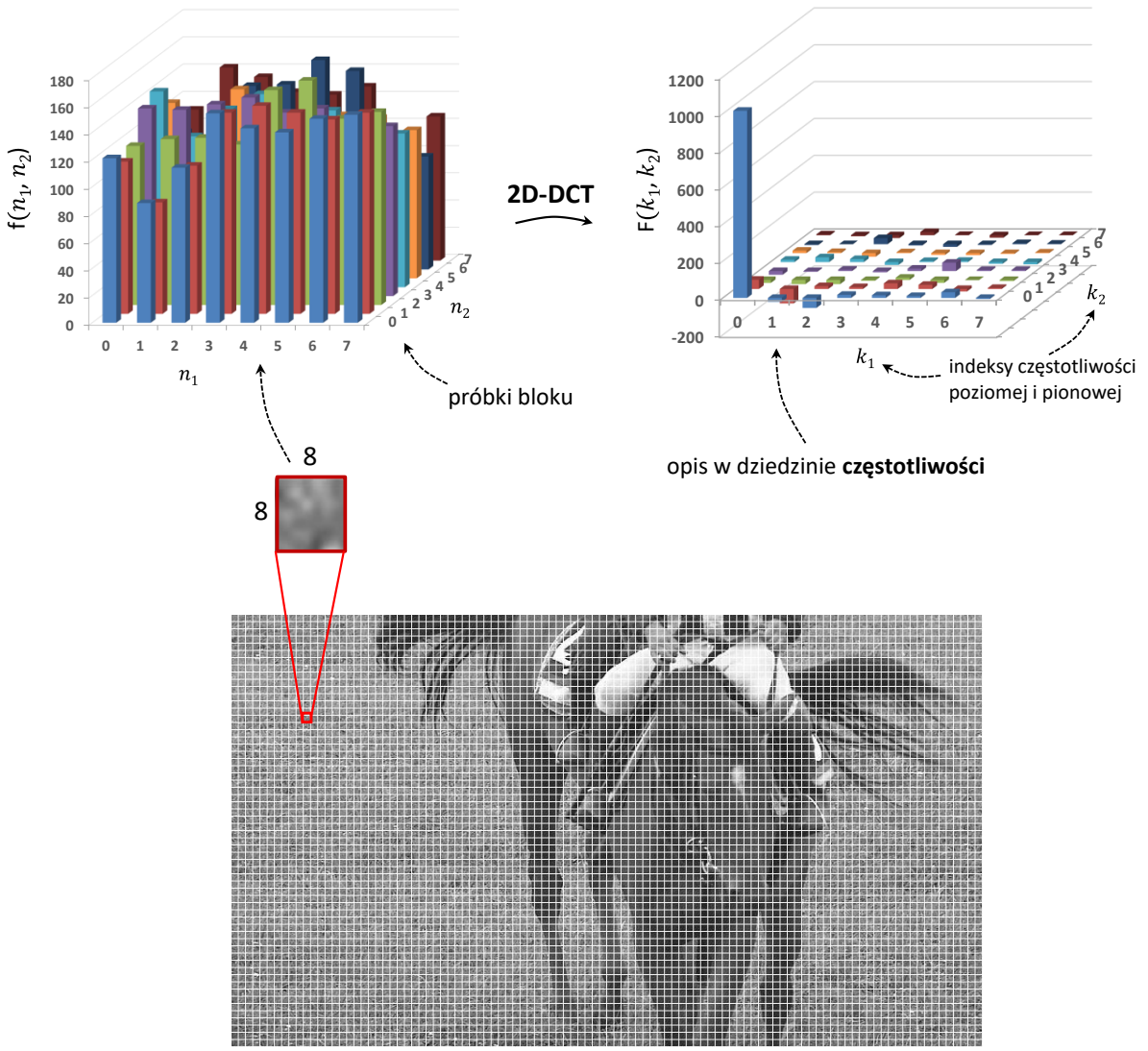
szczegółowych wyjaśnień w Rozdziale 2. W związku z powyższym, w tym miejscu przywołane zostaną jedynie najważniejsze wnioski z tamtej dyskusji:

1. Już mała liczba „niskoczęstotliwościowych” kosinusów pozwala z dużą dokładnością opisywać fragmenty obrazu, o prostej treści. Można to zaobserwować analizując zamieszczony na rysunku 4-17 wynik rozkładu na funkcje kosinusoidalne treści wybranego bloku 8x8 obrazu. W otrzymanym rezultacie najbardziej znaczące amplitudy posiadają kosinusy o niskich częstotliwościach. Należy zauważyć, że tych kosinusów jest bardzo niewiele, a pomimo tego przenoszą one znaczną część informacji o treści bloku obrazu. Przesłanie do dekodera tej (znaczącej) części informacji wiąże się z relatywnie małym kosztem bitowym.

2. Informacja, która opisuje składową (kosinusoidalną) o danej częstotliwości jest bardzo prosta. Składają się na nią tylko amplituda oraz znak składowej. Dodatkowego fazowego przesunięcia składowych nie trzeba już kodować, ponieważ są one z góry znane także w dekoderze obrazu. Tym samym, poinformowanie dekodera o obecności w treści bloku składowej (kosinusoidalnej) o danej częstotliwości wiąże się z bardzo niewielkim kosztem bitowym. Biorąc dodatkowo pod uwagę, że w przypadku znacznej części bloków obrazu tych kosinusów przesyła się do dekodera bardzo niewiele, to summa summarum skutkuje to bardzo wydajnym opisem treści obrazu.

3. Rozkład treści bloku na składowe kosinusoidalne jest obliczeniowo bardzo prosty. Wynika to z faktu istnienia szybkich algorytmów wyznaczania jednowymiarowych przekształceń DCT (1D-DCT) oraz tzw. **separowalności przekształcenia**, czyli możliwości wyznaczenia rezultatu przekształcenia dwuwymiarowego (2D-DCT) w drodze odpowiednich przekształceń 1D-DCT (więcej szczegółów na ten temat znaleźć można w Rozdziale 2).

Z wymienionych powyżej powodów opis danych obrazu z użyciem funkcji kosinusoidalnych zdecydowanie dominuje w koderach obrazu.



Rysunek 4-17. Rezultat przekształcenia 2D-DCT, zastosowanego dla bloku 8x8 próbek obrazu. Wynikiem przekształcenia jest informacja o amplitudzie oraz znaku dwuwymiarowych funkcji kosinusoidalnych, dzięki której możliwe jest opisanie treści bloku.

4.10.1.3. Kwantyzacja składowych kosinusoidalnych

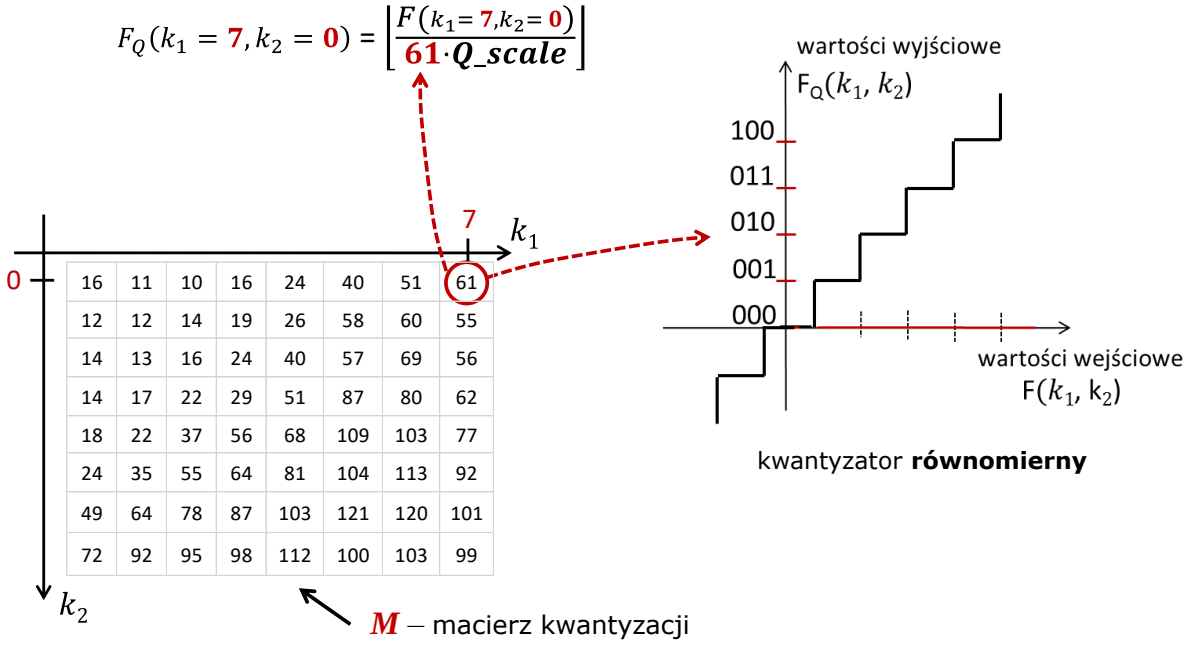
Ograniczona zdolność człowieka w zakresie percepcji składowych kosinusoidalnych otwiera możliwość ich przedstawiania w koderze z niepełną dokładnością. Dokładność reprezentacji składowych ogranicza się **kwantując** amplitudy poszczególnych kosinusów. Jak już wcześniej wskazano, kwantowanie to jest realizowane dzieląc amplitudy kosinusów przez określoną wartość liczbową.

Sposób kwantowania kolejnych składowych ma ogromne wręcz znaczenie z punktu widzenia uzyskiwanych rezultatów kodowania obrazu (rezultatów, rozumianych jako relacja pomiędzy liczbą bitów opisujących zakodowany obraz a jego jakością). W praktyce uwzględnia on (sposób kwantowania) obserwacje w zakresie zdolności i ograniczeń percepcji poszczególnych składowych przez widza. Zgodnie z tymi obserwacjami, człowiek wykazuje mniejszą czułość w przypadku „rejestracji” składowych wysokoczęstotliwościowych, a w miarę zmniejszania

częstotliwości składowych możliwości ich percepcji rosną. Daje to więc jasną wskazówkę, żeby poszczególne składowe kosinusowe (o danej wartości częstotliwości) kwantować w odmienny sposób.

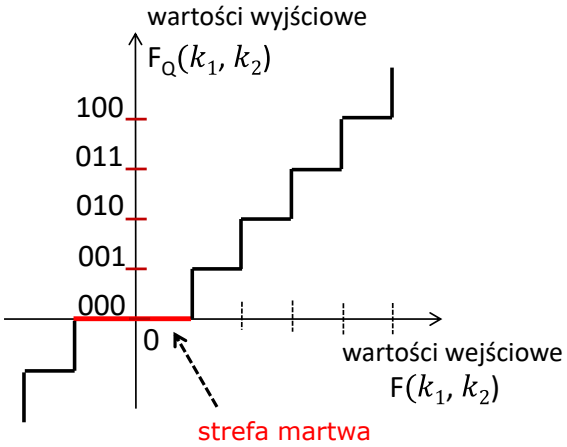
W koderach obrazu zróżnicowanie sposobu kwantowania poszczególnych „częstotliwości” uzyskuje się stosując tzw. **macierz (tablicę) kwantyzacji *M***. Poszczególne elementy tej macierzy zostały wyznaczone eksperymentalnie, uwzględniając charakterystyki czułości widza w zakresie postrzegania poszczególnych „częstotliwości”. Co do samej idei macierz ta mówi koderowi, przez jaką wartość liczbową należy podzielić amplitudę kosinusa o danej częstotliwości, żeby zrealizować wydajne jego kwantowanie (patrz Rysunek 4-18). Tak więc, w macierzy tej mamy większe wartości, przypisane dla wysokich „częstotliwości”, co odpowiada silniejszemu kwantowaniu tych składowych, oraz mniejsze wartości, które zostały skojarzone z niskimi „częstotliwościami”, co przekłada się na słabsze ich kwantowanie. Czyli uzyskujemy w efekcie taką reprezentację składowych, że te o niskich częstotliwościach są reprezentowane dokładniej, a składowe o wysokich częstotliwościach opisuje się z większym błędem.

Wspomniane podzielenie amplitudy składowej przez wartość macierzy kwantyzacji realizuje **równomierne kwantowanie** wartości amplitudy kosinusa (patrz Rysunek 4-18). Ponieważ w rozważanym przykładzie (kwantowanie w bloku 8x8 próbek DCT) macierz kwantyzacji zawiera 64 liczby, to można powiedzieć, że podczas kwantowania kosinusów wykorzystuje się 64 dedykowane kwantyzatory równomierne (bo w ogólności inne dla poszczególnych wartości częstotliwości), gdzie każdy z kwantyzatorów ma ściśle określoną szerokość przedziałów kwantyzacji.



Rysunek 4-18. Idea kwantowania składowych kosinusoidalnych z użyciem **macierzy (tablicy) kwantyzacji *M***. W przedstawionej formule matematycznej każda z liczb macierzy ***M*** realizuje równomierny kwantyzator, który jest przypisany składowej o danej częstotliwości. Dla bloku 8x8 próbek transformaty kosinusowej mamy więc 64 dedykowane kwantyzatory równomierne.

W praktyce, w wykorzystywanych kwantyzatorach równomiernych, zwiększa się jeszcze szerokość przedziału kwantowania wokół zerowej wartości wejściowej. Otrzymuje się w ten sposób tzw. **kwantyzator ze strefą martwą** (ang. dead zone), którego charakterystykę można zobaczyć na poniższym rysunku. **Strefa martwa** kwantyzatora pozwala na dodatkowe wyzerowanie części wejściowych amplitud kosinusów, które w przypadku zwykłego kwantyzatora bez tej strefy przyjęłyby po skwantowaniu bardzo małe wartości liczbowe (np. 1 lub 2). W takiej sytuacji błąd kwantowania i tak jest duży (w przypadku kwantyzatora ze strefą martwą lub bez niej), a dopuszczenie do dalszego kodowania tych małych wartości liczbowych dość zauważalnie zwiększyłoby rozmiar strumienia zakodowanych danych. Można więc powiedzieć, że użycie kwantyzatora z martwą strefą poprawia relację: stopień kompresji danych – jakość zakodowanego obrazu.



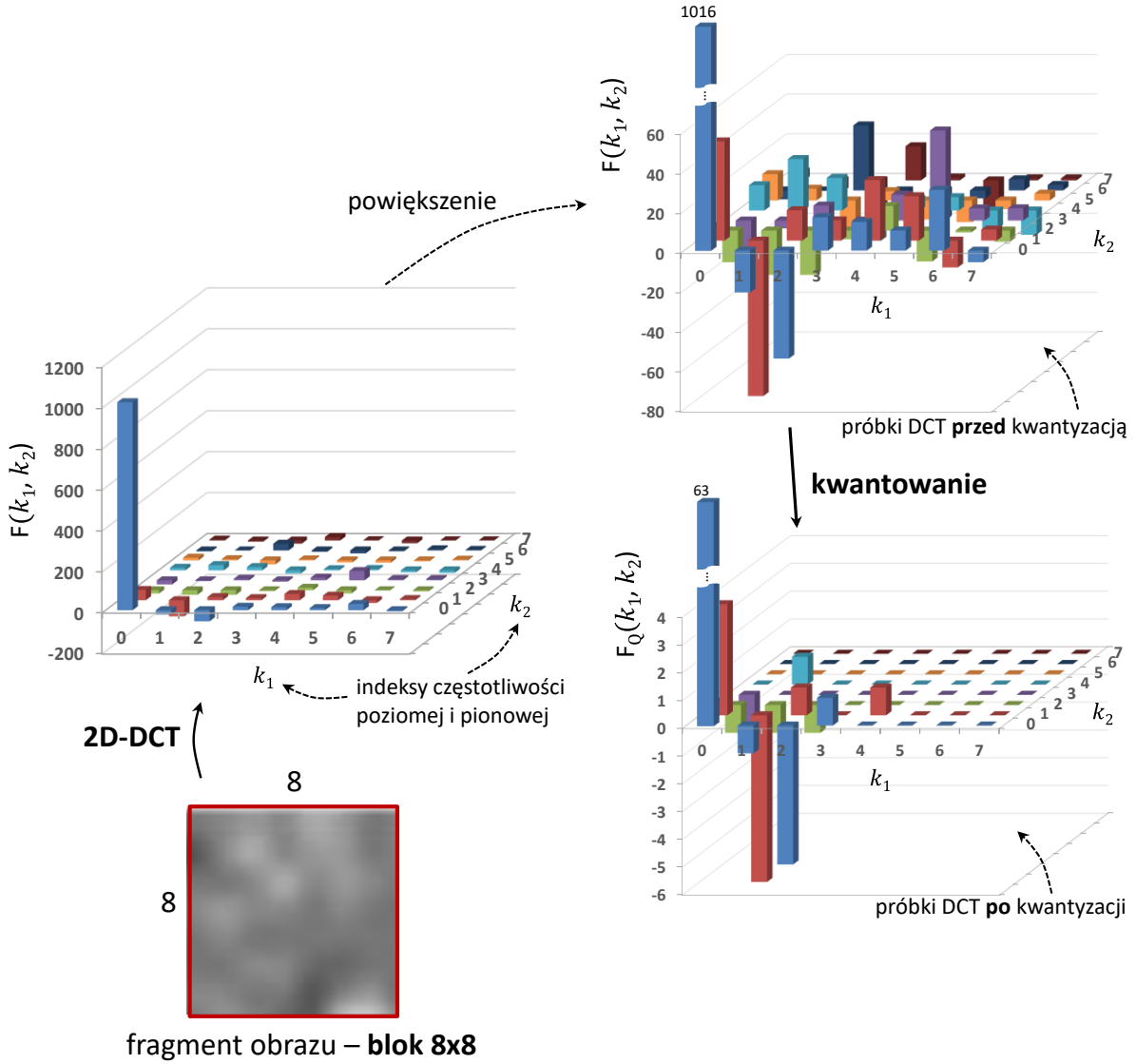
kwantyzator **równomierny** z **martwą strefą**

Rysunek 4-19. Równomierny kwantyzator z **martwą strefą**.

Powierzenie zadania kwantowania danych samej tylko macierzy kwantyzacji prowadziłby do z góry ustalonego rezultatu kodowania obrazu (z góry określone rozmiar strumienia zakodowanych danych oraz jakość zakodowanego obrazu). Użytkownik kodera, nie mógłby więc wpływać na proces kodowania obrazu. Jak wiadomo, kodery mogą być używane w zastosowaniach, o mocno różniących się parametrach, jak np. dopuszczalna prędkość bitowa zakodowanego strumienia danych (mała lub duża prędkość). Jest ona inna w telewizji cyfrowej niż np. w telefonii komórkowej. Dlatego w koderze obrazu jest dostępny mechanizm, który poza samą macierzą kwantyzacji, pozwala dodatkowo wpływać na sposób (siłę) kwantowania poszczególnych składowych kosinusowych. Jest to zamieszczony na rysunku 4-18 parametr **Q_scale**, który jest nazywany **współczynnikiem skali kwantyzatora**. Współczynnik ten skaluje wartości macierzy kwantyzacji, zmieniając tym samym sposób kwantowania poszczególnych kosinusów. Przy takim podejściu, dla danego kwantyzatora, faktyczna szerokość przedziałów kwantyzacji (kroku kwantowania) jest więc iloczynem określonej wartości macierzy kwantyzacji oraz współczynnika Q_scale. Z użyciem parametru Q_scale krok kwantowania składowych kosinusoidalnych może być zmieniany na poziomie bloku obrazu (bo co blok obrazu możemy zmieniać wartość tego parametru). Współczynnik ten daje więc możliwość sterowania procesem kodowania obrazu w bardzo szerokim zakresie (od kompresji silnej do bardzo słabej).

Analizując same już wartości liczb tablicy kwantyzacji widzimy, jak silnie podczas kwantowania mogą zostać zmniejszone amplitudy poszczególnych kosinusów. W przypadku

tablicy kwantyzacji z rysunku 4-18 (czyli zakładając, że **Q_scale** = 1) byłaby to wielokrotna redukcja wartości amplitudy! Rezultatem kwantowania są zatem w ogólności małe wartości amplitud kosinusów, a w przypadku kosinusów o wysokich częstotliwościach przestrzennych kwantowanie prowadzi wręcz do ich całkowitego usunięcia z treści kodowanego obrazu. Odpowiada to usuwaniu z obrazu części informacji o kodowanej treści. Przytoczone wnioski można łatwo zaobserwować, porównując zamieszczone na rysunku 4-20 wartości próbek przekształcenia 2D-DCT przed oraz po kwantowaniu. Usuwana na etapie kwantowania część informacji jest bezpowrotnie tracona. Powoduje to określoną utratę jakości kodowanego obrazu, ale jest w koderze źródłem ogromnych oszczędności bitowych reprezentacji danych.

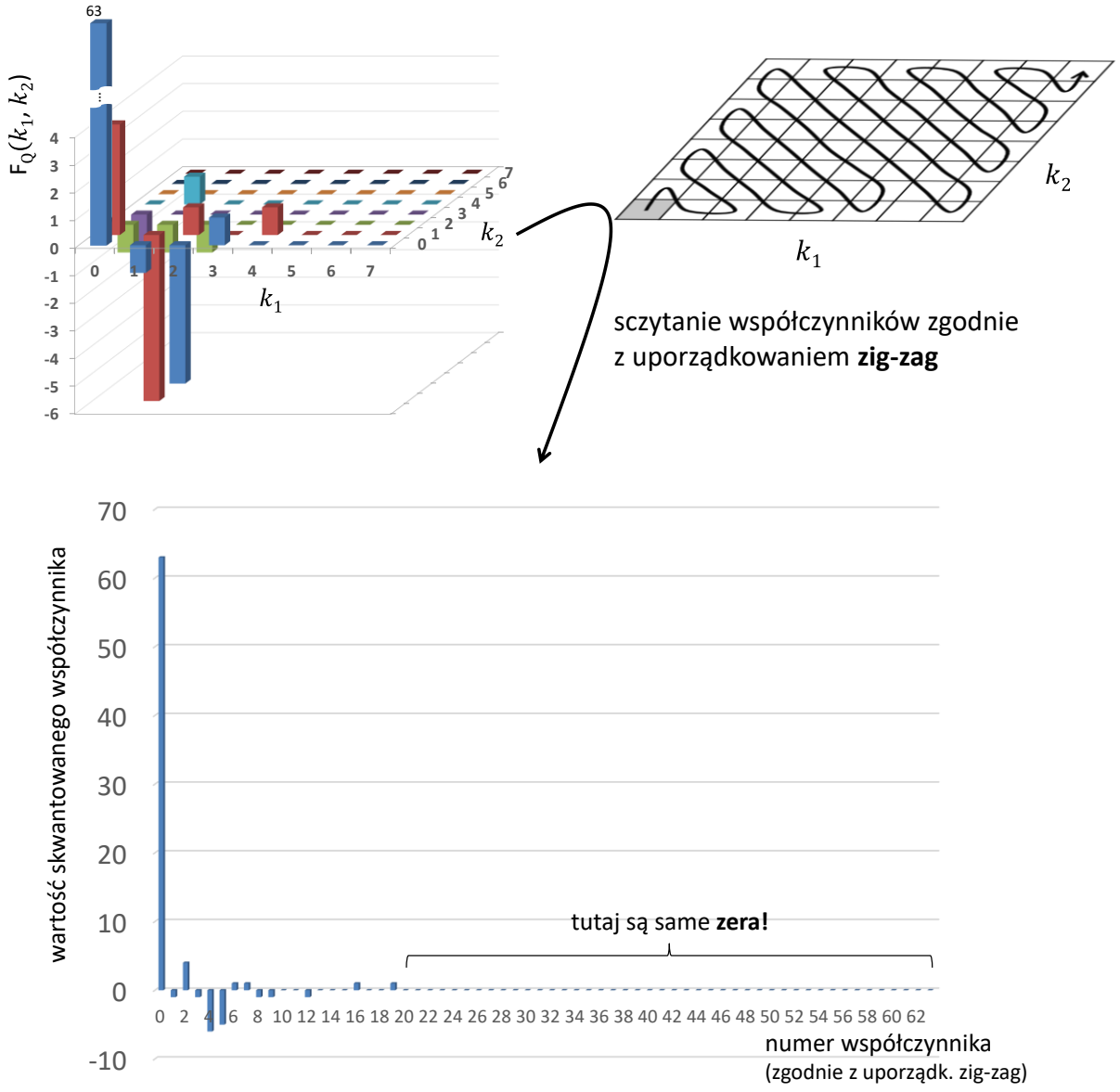


Rysunek 4-20. Kwantyzacja składowych kosinusoidalnych, które są wynikiem przekształcenia 2D-DCT bloku 8x8 próbek obrazu. Kwantyzacja silnie zmniejsza wartość amplitudy poszczególnych kosinusów, przez co redukuje dokładność ich reprezentacji w koderze obrazu.

4.10.1.4. Szeregowanie skwantowanych kosinusów

W wyniku kwantowania składowych duża część kosinusów wysokoczęstotliwościowych jest zazwyczaj całkowicie zerowana przez koder. Po kwantowaniu, zwykle niezerowe są więc tylko amplitudy składowych o najniższych częstotliwościach przestrzennych.

Z uwagi na wspomniany charakter wartości amplitud poszczególnych składowych są one szeregowane zgodnie z tzw. **uporządkowaniem zig-zag**. W tym uporządkowaniu, amplitudy kosinusów (wraz ze znakiem) są odczytywane z dwuwymiarowej tablicy w ściśle określonej kolejności (przedstawionej na poniższym rysunku), i umieszczane na kolejnych pozycjach tablicy jednowymiarowej. Nową, jednowymiarową tablicę, rozpoczyna więc amplituda (wraz ze znakiem) kosinusa o zerowych częstotliwościach przestrzennych poziomej i pionowej (tzw. **współczynnik DC** przekształcenia kosinusowego), po którym następują 63 współczynniki opisujące pozostałe składowe (tzw. **współczynniki AC**, z co najmniej jedną niezerową częstotliwością przestrzenną – poziomą lub pionową). W wyniku takiego uszeregowania danych uzyskuje się znaczące skupienie obok siebie wartości niezerowych amplitud (wraz z towarzyszącą informacją o znaku), a tym samym znaczne zgrupowanie wartości zerowych. Zwykle, tylko początkowa część jednowymiarowej tablicy zawiera niezerowe wartości amplitud. Takie ułożenie danych otwiera możliwość wydajnej ich reprezentacji w koderze, co stanowi temat kolejnego punktu.



Rysunek 4-21. Przepisanie amplitud (wraz ze znakiem) kosinusów z dwuwymiarowego bloku do jednowymiarowej tablicy. Przepisanie współczynników jest realizowane z zachowaniem ściśle określonej kolejności, określanej w literaturze mianem **uporządkowania zig-zag**.

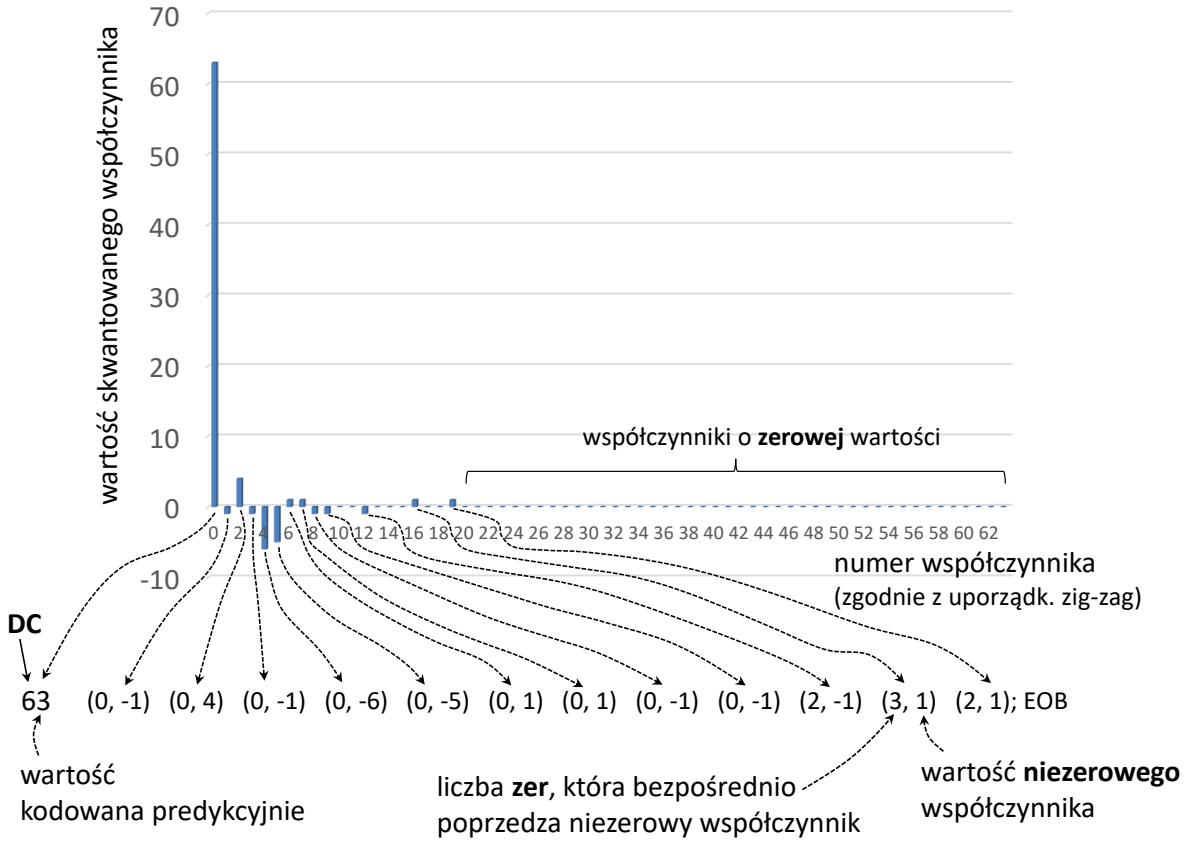
4.10.1.5. Kodowanie skwantowanych amplitud kosinusów

Kodowanie amplitud rozpoczyna się od właściwej reprezentacji współczynnika DC. Jak wiadomo, współczynnik ten odzwierciedla sumę wartości wszystkich próbek bloku obrazu (patrz wzory dotyczące przekształcenia 2D-DCT). Z uwagi na silne podobieństwo treści sąsiednich bloków w obrazie sumy próbek tych bloków będą także podobne. Dlatego współczynniki DC kolejnych bloków obrazu warto jest kodować predykcyjnie, a nie zupełnie niezależnie od siebie. Współczynnik DC danego bloku jest więc przewidywany w oparciu o znane wartości tego współczynnika w przetworzonych (zakodowanych bądź zdekodowanych) już blokach obrazu, które bezpośrednio z nim sąsiadują. W najprostszym przypadku może to być przewidywanie na podstawie jednej, ostatnio przetworzonej wartości DC. Wysoki stopień podobieństwa sąsiadujących ze sobą współczynników DC skutkuje (zwykle) małymi błędami przewidywania

wartości, przez co sygnał błędu predykcji lepiej poddaje się kompresji niż oryginalne liczby, które wprost wyrażają kolejne składowe DC.

Po współczynniku DC następuje określona liczba niezerowych współczynników AC, między którymi znajdują się ciągi zerowych wartości. Tego typu dane można wydajnie reprezentować stosując metodę **kodowania (długości) ciągów** (ang. run-length encoding – RLE). Metoda ta pozwala na reprezentowanie w sposób łączny ciągu wielu zer, razem z niezerową wartością, która za tym ciągiem występuje. Robi to przy pomocy zaledwie dwóch liczb: pierwszej określającej liczbę zer w ciągu i drugiej, która wyraża wartość niezerowego współczynnika, który występuje za tym ciągiem. Niezerowy współczynnik, wraz z poprzedzającym go ciągiem zer są więc reprezentowane parą liczb. Idea metody kodowania ciągów zostało dodatkowo zilustrowana na rysunku 4-22.

Należy dodatkowo podkreślić, że patrząc od końca, dużą część tablicy (tej, która gromadzi współczynniki po uszeregowaniu zig-zag) wypełniają same zera. Nie są one oczywiście kodowane i przesyłane do dekodera jedno po drugim. Zamiast tego, koder wykrywa w tablicy położenie ostatniego niezerowego współczynnika transformaty, żeby pozostałe zera zakodować jednym, bardzo krótkim słowem kodowym, określanym jako **EOB** (ang. end of block).



Rysunek 4-22. Kodowanie (długości) ciągów, czyli wydajny sposób reprezentacji ciągu kolejnych zer, oraz niezerowej wartości, która występuje za tym ciągiem.

4.10.1.6. Kodowanie entropijne danych

Wartości błędu przewidywania współczynników DC, czy pary liczb, które opisują niezerowe wartości AC oraz znajdujące się między tymi wartościami zera ciągle wykazują pewną nadmiarowość statystyczną. Ostatnim etapem kodowania danych jest dalsza redukcja tej nadmiarowości, poprzez zastosowanie **kodowania entropijnego** przywołanych danych. Kodowanie entropijne przypisuje ciągi bitów kodowanym wartościom, których długość jest bezpośrednio uzależniona od prawdopodobieństwa ich wystąpienia w strumieniu kodowanych danych.

W starszych koderach obrazu zastosowanie znajdują proste realizacje entropijnej kompresji danych. Powszechnie stosuje się metodę kodowania **Huffmana**, czy **kodowanie arytmetyczne** danych, jednak stosowany na potrzeby tych metod sposób estymacji statystyki danych jest bardzo, ale to bardzo uproszczony. A jak wiadomo, wydajność całej techniki kompresji entropijnej zależy w ogromnym stopniu od tego właśnie elementu. Powszechnie stosuje się prostą wiedzę o statystyce danych, wyliczoną wcześniej na bazie zbioru obrazów testowych, bez możliwości uaktualniania tych statystyk w trakcie kodowania danych obrazu. Oczywiście, te proste rozwiązania ograniczają efektywność kompresji danych, jednak z drugiej strony redukują ponoszone w koderze i dekoderze nakłady obliczeniowe.

4.10.1.7. Wydajność prostych metod kompresji transformatowej

4.10.1.7.1. Efektywność kompresji

Wydajność kodowania zostanie oceniona analizując rezultaty kompresji stratnej obrazu, czyli jakość obrazu po kompresji oraz współczynnik kompresji (stopień kompresji) danych, a więc stopień redukcji zbioru danych, który opisuje obraz. Żeby móc taką analizę przeprowadzić dokonano kompresji wybranego obrazu testowego z wykorzystaniem dość dobrze znanego kodera **„mozjpeg”**. Koder ten stanowi wysokowydajną, z punktu widzenia osiąganej efektywności kompresji danych, realizację techniki kodowania JPEG.

Poniższe rysunki prezentują kolejne wyniki kompresji obrazu, jakie otrzymano w trakcie testu, przy założeniu określonej jakości obrazu po kompresji. Są to wyniki kodowania, które odpowiadają subiektywnej jakości obrazu dekodowanego w dekoderze od bardzo dobrej do bardzo słabej. Celem właściwej oceny poziomu zniekształceń w zakodowanych obrazach, warto jest je oglądać w odpowiednim powiększeniu. Dla porównania, na pierwszym rysunku zamieszona została oryginalna wersja obrazu testowego, który nie był wcześniej poddawany kompresji. Obraz ten jest pierwszym obrazem sekwencji wizyjnej „PartyScene”.

Rysunek 4-23. Pierwszy obraz sekwencji „PartyScene”. Obraz **oryginalny, nieskompresowany**.

Rysunek 4-24. Zdekodowany obraz po wcześniejszej kompresji. Stopień kompresji wynosi **16,58.**

Rysunek 4-25. Zdekodowany obraz po wcześniejszej kompresji. Stopień kompresji wynosi **22**.

Rysunek 4-26. Zdekodowany obraz po wcześniejszej kompresji. Stopień kompresji wynosi **34,53.** W tym przypadku zaczynają być już widoczne zniekształcenia w niektórych fragmentach obrazu.

Rysunek 4-27. Zdekodowany obraz po wcześniejszej kompresji. Stopień kompresji wynosi **90,53.** Widoczne duże zniekształcenia w przeważającej części obrazu.

ł

Rysunek 4-28. Zdekodowany obraz po wcześniejszej kompresji. Stopień kompresji wynosi **155,95.** Bardzo, bardzo silne zniekształcenia treści w całym obrazie.

Zgodnie z otrzymanymi wynikami bardzo dobrą jakość zakodowanego obrazu uzyskuje się realizując kodowanie ze współczynnikiem kompresji na poziomie kilka – kilkanaście. Od wartości 20 (mowa o współczynniku, czyli stopniu kompresji) zaczynają się już na obrazie pojawiać

widoczne zniekształcenia kompresji (np. mało intensywny **efekt blokowy** – patrz punkt 4.10.2.4.1, czy zniekształcenia widoczne na krawędziach obiektów), ale sytuacja taka ma miejsce w pojedynczych tylko fragmentach obrazu. W przypadku stopnia kompresji powyżej 30 liczba takich fragmentów w obrazie szybko rośnie. Powyżej stopnia o wartości 50 wspomniane zniekształcenia zaczynają już obejmować znaczącą część kodowanej treści obrazu i są już (bardzo) dokuczliwe. W tym przypadku istnieje już wrażenie słabej, bądź bardzo słabej jakości obrazu po kompresji. Gdyby zmusić koder obrazu do jeszcze silniejszej kompresji danych, np. kompresji więcej niż 100 krotnej, to jakość otrzymanego w ten sposób obrazu po kompresji będzie już fatalna.

Należy również mocno podkreślić, że treść kodowanego obrazu ma wpływ na rezultaty kompresji (stopień kompresji – jakość zakodowanego obrazu), jednak zdaniem autora przedstawione w tym punkcie przykładowe wyniki kodowania dobrze oddają faktyczne możliwości (jakby nie patrzeć to jednak skromne) prostych koderów transformatowych obrazu.

4.10.1.7.2. Złożoność obliczeniowa

Pozytywną stroną prostych technik kompresji jest bardzo niewielka złożoność kodowania i dekodowania obrazu. Przekłada się to oczywiście na krótki czas kompresji i dekompresji danych. W przypadku zastosowanego w tym punkcie kodera (koder **„mozjpeg”**) 16 krotna kompresja danych obrazu o wysokiej rozdzielczości (rozdzielczość przestrzenna 1920x1080) wiązała się z czasem kodowania obrazu na poziomie 280 ms, przy czym jego dekompresja zabierała procesorowi już tylko 7,5 ms. Wraz ze wzrostem stopnia kompresji danych te czasy się jeszcze skracają. I tak np. kompresja 30 krotna to czasy kodowania i dekodowania obrazu odpowiednio 140 ms oraz 5 ms. Kompresja 40 krotna to już 115 ms i 4,5 ms (czasy kodowania i dekodowania). A 80 krotna kompresja to jeszcze krótsze czasy – 80 ms i 4ms. Takie czasy otrzymano wykonując program kodeka na procesorze **Intel Core i7 – 4770**, taktowanego zegarem **3,4GHz**. Według autora, skrócenie czasów kodowania i dekodowania obrazu, które obserwuje się wraz ze wzrostem stopnia kompresji danych, może głównie wynikać ze zmniejszającej się na tej drodze objętości danych, które są później w kodeku (koder i dekoder) przedmiotem entropijnego kodowania (dekodowania).

Oprogramowanie **„mozjpeg”** techniki JPEG nie należy oczywiście do najszybszych. Koder ten realizuje złożoną obliczeniowo metodę kodowania danych, która w taki sposób optymalizuje kwantowanie próbek widma bloków obrazu, żeby uzyskać na tej drodze możliwie najlepszą relację pomiędzy prędkością bitową strumienia zakodowanych danych a jakością obrazu po kompresji[39]. Jak się okazuje użycie takiego mechanizmu zwiększa o **kilka – kilkanaście** procent (typowe wartości to 4% – 15%) efektywność kompresji obrazu, jednak wydłuża czas kodowania nawet kilka razy. Rezygnując więc w koderze ze stosowania wspomnianej optymalizacji, oraz dodatkowo wykonując niektóre obliczenia z wykorzystaniem instrukcji współczesnych procesorów (np. zestaw instrukcji Streaming SIMD Extensions – SSE, który pozwala na jednoczesne przetwarzanie wektora liczb, zamiast sekwencyjnego operowania na pojedynczych wartościach), możliwe jest znaczne przyspieszenie działania kodera. Dobrym tego przykładem jest oprogramowanie **„libjpeg-turbo”** kodera JPEG, które jest przeciętnie 10 razy szybsze od kodera **„mozjpeg”.** Użycie więc tego kodera w scenariuszu 25-krotnej kompresji obrazu pozwala na kodowanie aż 60 obrazów o rozdzielczości ‘full HD’ w czasie 1 sekundy (na przywołanym powyżej procesorze). Należy w tym miejscu wyraźnie podkreślić, że ta prędkość kodowania obrazów będzie jeszcze większa w przypadku sprzętowych koderów obrazu (dedykowany układ scalony). Niska złożoność kodera i dekodera obrazu skłaniają do praktycznego użycia prostych technik kompresji

[39] Oczywiście w kontekście użycia algorytmów kompresji, jakie mają zastosowanie w technice JPEG.

również dzisiaj (w niektórych zastosowaniach), jak np. kodowanie obrazu w urządzeniach, które działają na baterii.

4.10.2. Kodowanie transformatowe w wydaniu zaawansowanym

Znakomitym przykładem zaawansowanej realizacji idei kompresji transformatowej jest technika HEVC (ang. High Efficiency Video Coding) kodowana obrazu, która została opracowana po roku 2010 [HEVC, HEVC rec][40]. Ten nowy koder realizuje bardzo złożone algorytmy kompresji danych, które są osiągnięciem badań naukowych z ostatnich kilkunastu lat[41]. Koder HEVC realizuje bardzo wydajną kompresję ruchomego obrazu, jednak z jego użyciem można zakodować również obraz statyczny, co będzie przedmiotem zainteresowania w tym punkcie.

W stosunku do prostych koderów obrazu w najnowszych rozwiązaniach można zaobserwować następujące, kluczowe zmiany:

1. Idea kompresji transformatowej (czyli ciąg operacji: DCT + kwantowanie próbek widma + właściwe ich kodowanie) nie jest, tak jak w starszych koderach, bezpośrednio realizowana na próbkach obrazu. Tutaj, najpierw dokonuje się przewidywania wartości kolejnych próbek obrazu, żeby kodować już tylko błąd przewidywania próbek, czyli **sygnał błędu predykcji** próbek obrazu.

2. Predykcja wartości próbek, jak również właściwe kodowanie transformatowe błędu przewidywania tych próbek są dokonywane w kolejnych fragmentach obrazu, które są blokami, będące z reguły kwadratami. Starsze kodery dzieliły obraz na bloki o jednakowym, z góry ustalonym rozmiarze (były to np. bloki o rozmiarze 8x8 próbek). Najnowsze kodery (w tym HEVC) dzielą obraz nierównomierną siatką bloków, w której rozmiary poszczególnych bloków mogą się różnić między sobą, i są na przykład wielkości od 4x4 do 64x64. Uwzględniając charakter treści aktualnie kodowanego fragmentu obrazu koder może zatem wybrać właściwe rozmiary dla poszczególnych bloków, które pozwolą na uzyskanie wysokiej efektywności kompresji danych.

3. Decyzja o rozmiarze bloku jest niezależnie podejmowana dla etapów predykcji próbek oraz samego kodowania transformatowego danych.

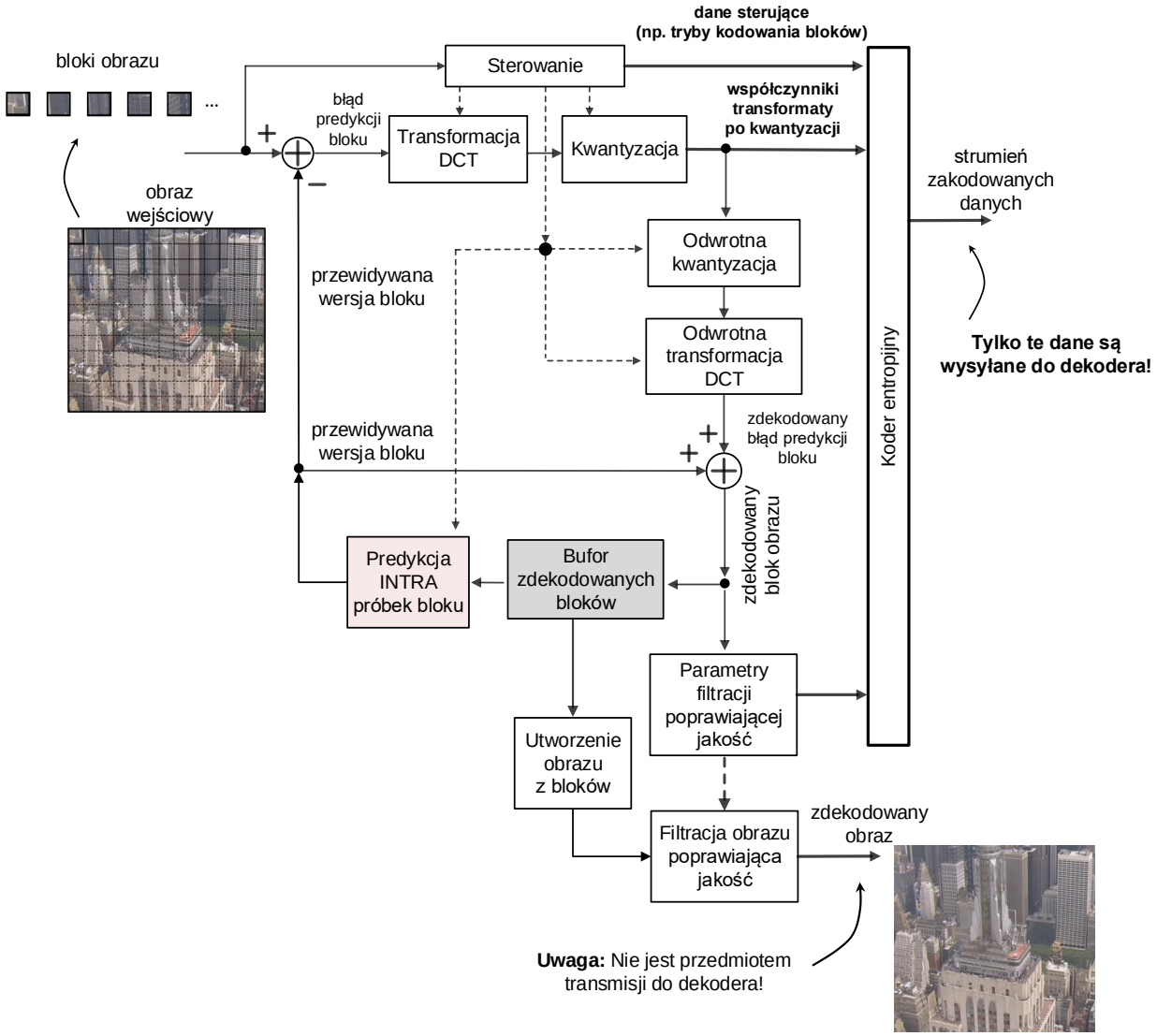
4. Niezależnie od podjętej decyzji o rozmiarze bloków obrazu koder dysponuje obszerną paletą sposobów przewidywania próbek, które wchodzą w skład bloków. W przypadku kodera HEVC tych sposobów jest aż 35, przy czym tak duża liczba trybów predykcji próbek jest dostępna w przypadku każdego dopuszczalnego rozmiaru bloku.

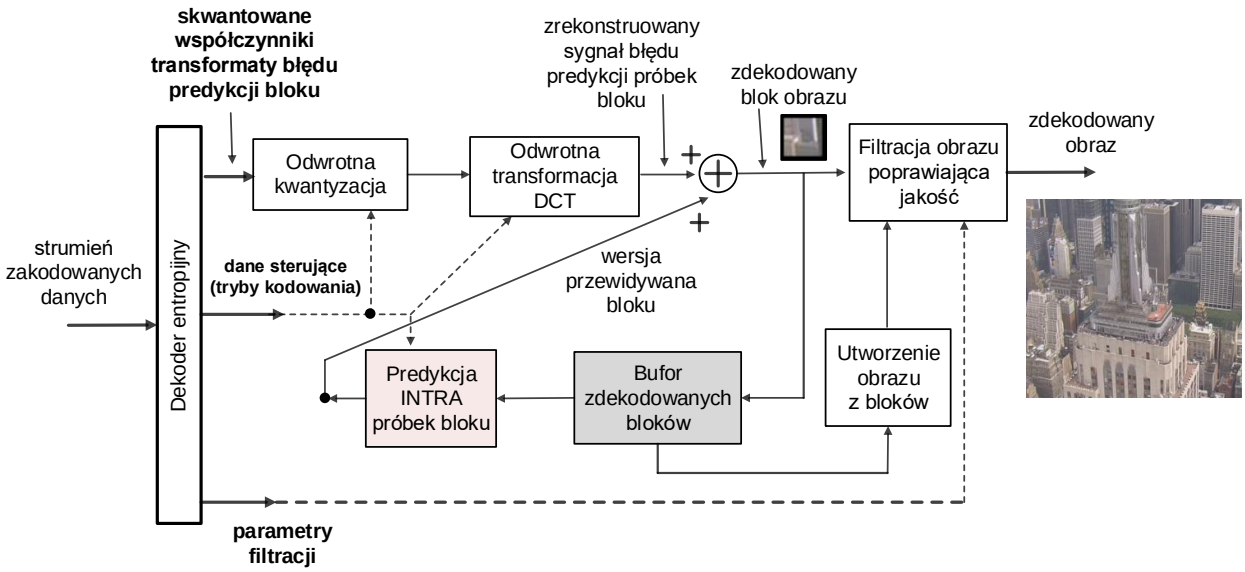
5. Stratna kompresja obrazu może w niektórych przypadkach doprowadzić do wystąpienia widocznych zniekształceń treści obrazu, czyli artefaktów kodowania. Ich obecność pogarsza oczywiście jakość skompresowanego obrazu. Żeby tę jakość poprawić, w najnowszych kodekach (w tym w HEVC) przeprowadza się filtrację obrazu, która redukuje artefakty kompresji. Warto tutaj podkreślić, że filtracji poprawiającej jakość nie stosowały starsze kodeki obrazu, np. JPEG czy MPEG-2.

[40] Mowa jest tutaj tylko o zdefiniowanych w technice HEVC narzędziach kompresji wewnątrzobrazowej.
[41] Algorytmy techniki HEVC (przynajmniej duża ich część) zostały opracowane po roku 2005, jednak pracując nad nową techniką korzystano oczywiście z osiągnięć naukowych lat wcześniejszych.

6. Pozostałe narzędzia kompresji danych, które są również obecne w starszych koderach obrazu (np. kodowanie entropijne danych) są w najnowszych koderach dużo bardziej zaawansowane, a przez to cechują się znacznie wyższą wydajnością.

Wymienione ulepszenia kompresji istotnie komplikują działanie kodera oraz dekodera. Można się już o tym przekonać porównując blokowe schematy kodeków starszego (rysunek 4-15) oraz tego współczesnego (rysunek 4-29). Jednak ulepszenia te są również źródłem dużo wyższej niż w koderach starszych, efektywności kompresji obrazu. Poszczególne elementy współczesnego kodeka zostaną w bardziej szczegółowy sposób omówione w kolejnych punktach.





Rysunek 4-29. Schemat blokowy **zaawansowanego** kodeka transformatowego obrazu (koder – schemat powyżej + dekoder – schemat poniżej). Schemat ten odzwierciedla sposób kompresji obrazu we współczesnym kodeku HEVC (tylko tryb wewnątrzobrazowy).

4.10.2.1. Przewidywanie treści obrazu

Ten element współczesnego kodera stanowi zasadniczą nowość w porównaniu ze starszymi technikami kompresji. Pozwala on bowiem na silne uproszczenie postaci danych (poprzez redukcję nadmiarowości przestrzennej próbek obrazu naturalnego), które są następnie przedmiotem kompresji transformatowej.

Sposób, w jaki jest przewidywana treść poszczególnych fragmentów obrazu ma ogromne wręcz znaczenie dla uzyskiwanych wyników kompresji danych. To, jak najlepiej przewidywać treść obrazu, żeby było ono obarczone małym błędem i umożliwiło tym samym wysoką efektywność kompresji danych było już przedmiotem dyskusji w punkcie 4.5. Współczesne kodery obrazu w pełni realizują wymienione tam zasady dokonując przewidywania treści w następujących krokach:

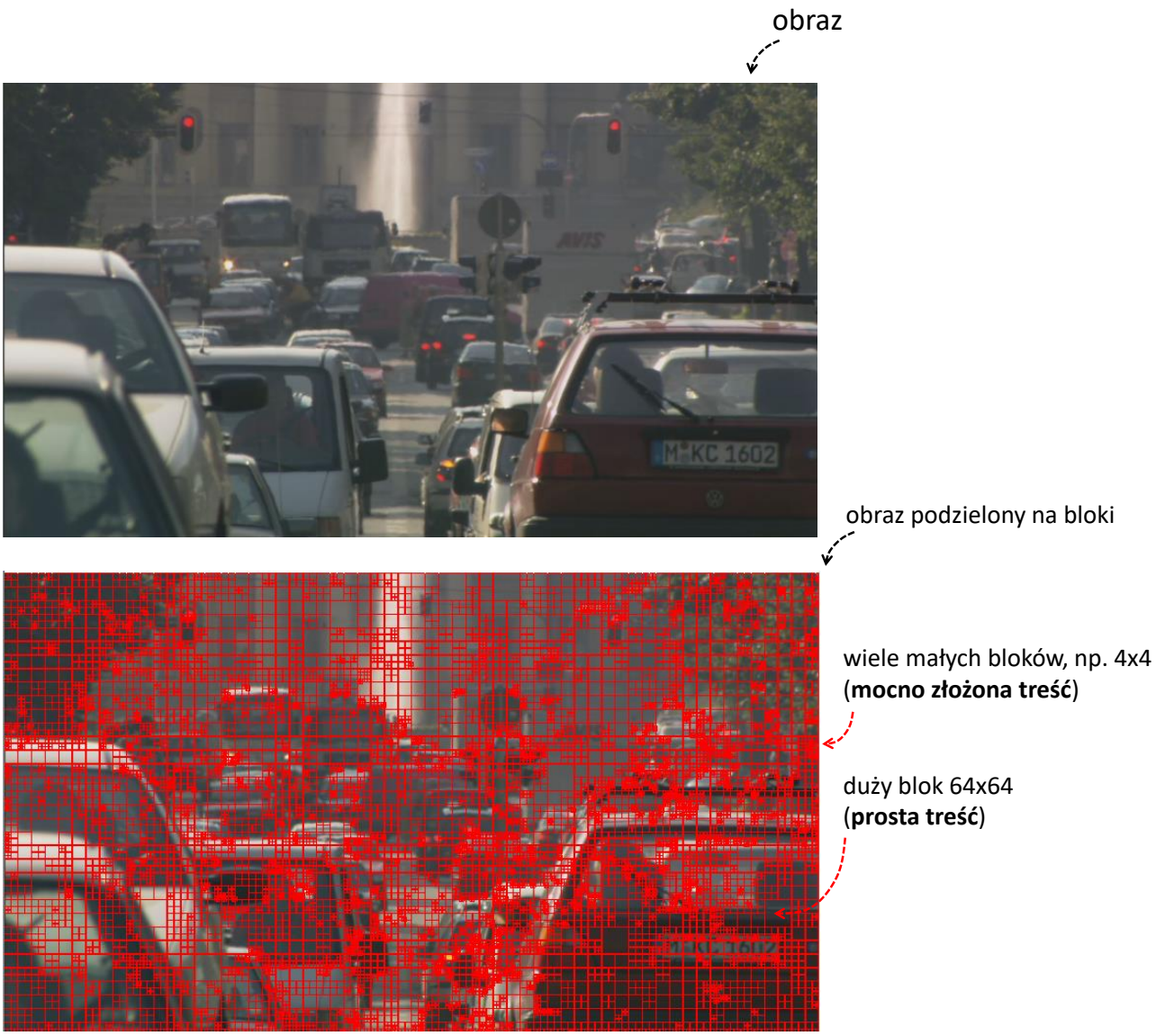
1. Najpierw obraz jest dzielony na bloki o ustalonym rozmiarze. Dany blok może mieć rozmiar (tak jest w przypadku kodera HEVC): 4x4, 8x8, 16x16, 32x32, 64x64[42]. Rozmiary poszczególnych bloków są arbitralnie ustalane przez koder uwzględniając charakter treści, która znajduje się w danym fragmencie obrazu. W tym miejscu koder kieruje się generalną zasadą: prosta treść – większe bloki, złożona, skomplikowana treść – bloki mniejsze. Przykładowy, rzeczywisty rezultat podziału obrazu na bloki o zmiennym rozmiarze został zaprezentowany na rysunku 4-30.

2. Następnie koder przewiduje treść każdego z bloków. Próbki bloku są przewidywane na podstawie odtworzonych (zdekodowanych!) już wcześniej wybranych próbek innych

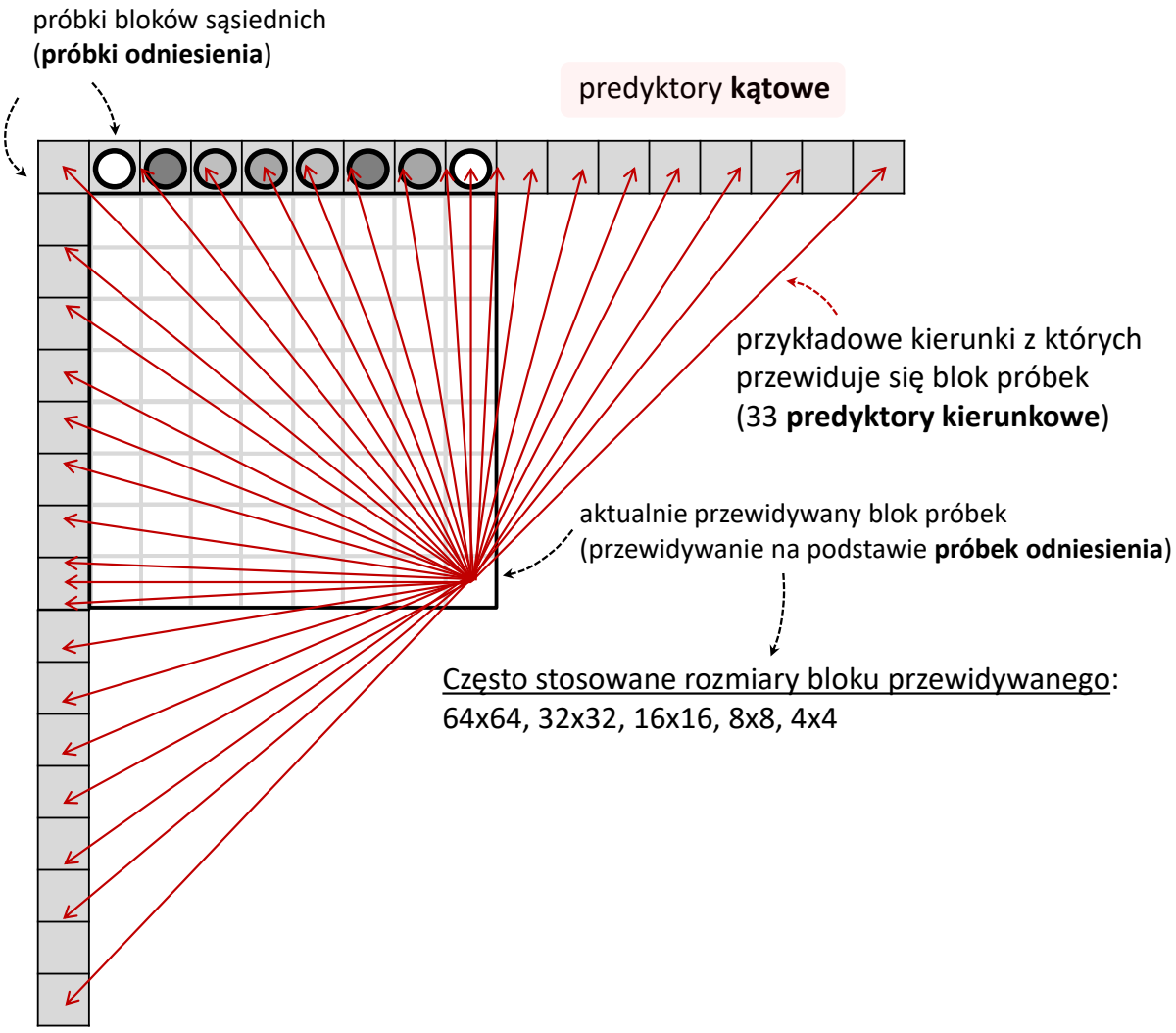
[42] W koderze HEVC, predykcja treści bloku o wielkości 64x64 przebiega trochę nietypowo. Zamiast dokonywać predykcji całego takiego bloku, to blok ten dzieli się na 4 mniejsze bloki o rozmiarze 32x32 i przewiduje się treść tych mniejszych bloków. Jednak dla każdego z tych 4 bloków stosuje się ten sam predyktor wartości próbek, przez co tryb predykcji jest sygnalizowany tylko raz (na poziomie bloku 64x64). Proszę zauważyć, że jest to inny wariant predykcji treści niż ten, w którym w każdym z bloków 32x32 stosuje się odmienny predyktor treści (co wymaga sygnalizacji trybu cztery razy, bo na poziomie bloków 32x32).

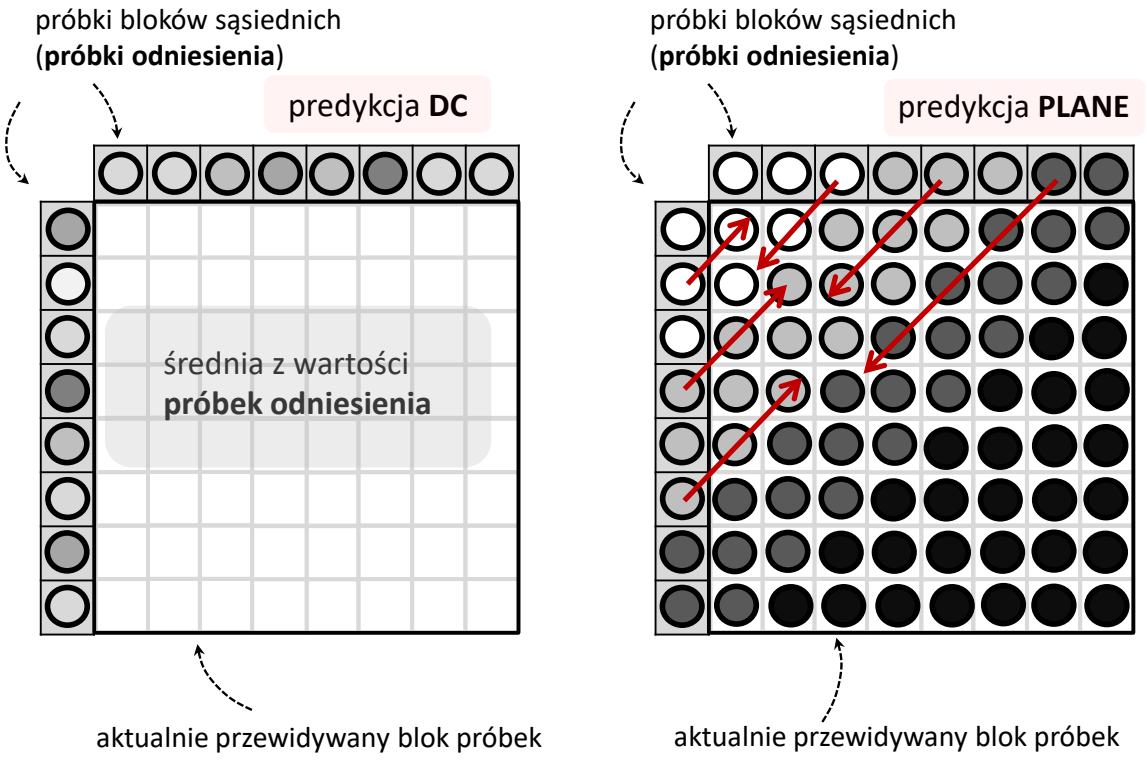
bloków, które bezpośrednio sąsiadują z aktualnie przetwarzanym blokiem (patrz rysunek 4-31). Próbki obrazu, które są podstawą predykcji innych próbek będą w tym miejscu określane **próbkami odniesienia**. W przypadku najnowszych koderów obrazu odniesieniem dla predykcji są próbki, które w stosunku do aktualnie przetwarzanego bloku znajdują się powyżej lub po lewej stronie, ale również próbki z bloków: górny – prawy, dolny – lewy oraz próbka bloku górnego – lewego. Analizując stopień złożoności oraz rodzaj treści w aktualnie przetwarzanym bloku koder dokonuje jego predykcji z użyciem jednego z kilkudziesięciu dostępnych sposobów predykcji treści (predyktorów próbek). W przykładowym koderze HEVC tych sposobów jest aż 35, i wszystkie one dostępne są dla każdego rozmiaru bloku (od 4x4, poprzez 8x8, do 64x64 – patrz rysunek 4-31). Ponieważ w obrazach naturalnych treść bloku może być: prostą teksturą, fragmentem z kierunkowymi krawędziami obrazu, czy obszarem o liniowej, płynnie zmieniającej się jasności próbek, to pula stosowanych predyktorów treści obejmuje między innymi: predyktor składowej stałej (tzw. predyktor DC), dużą liczbę predyktorów kątowych (obsługujące zmiany treści, które odbywają się pod określonym kątem) oraz predyktor PLANE, który jest dedykowany do wydajnego przewidywania treści o liniowo zmieniającej się jasności próbek.

Należy w tym miejscu mocno podkreślić, iż szeroki wachlarz możliwych rozmiarów bloków, jak również dostępność dużej liczby predyktorów treści tych bloków, dają koderowi możliwość wyboru najlepszej kombinacji **rozmiar bloku**, **predyktor treści** bloku, która dla danego fragmentu obrazu umożliwia uzyskanie najlepszych wyników predykcji treści.



Rysunek 4-30. Obraz testowy „Rush Hour” oraz rezultat podziału tego obrazu na bloki o określonym rozmiarze. Otrzymana siatka bloków jest wynikiem podjętych przez modelowe oprogramowanie HM 10.0 kodera HEVC decyzji o sposobie kodowania treści obrazu. W tych blokach dokonywana jest predykcja próbek obrazu. Przykład został opracowany z użyciem oprogramowania analizatora HEVC, którego głównym wykonawcą jest Pan Piesik Daniel.


próbki bloków sąsiednich
(**próbki odniesienia**)
predyktory **kątowe**
przykładowe kierunki z których
przewiduje się blok próbek
(33 **predyktory kierunkowe**)
aktualnie przewidywany blok próbek
(przewidywanie na podstawie **próbek odniesienia**)
Często stosowane rozmiary bloku przewidywanego:
64x64, 32x32, 16x16, 8x8, 4x4



Często stosowane rozmiary bloku przewidywanego:
64x64, 32x32, 16x16, 8x8, 4x4

Rysunek 4-31. Przykładowy zbiór predyktorów wartości próbek, które znajdują zastosowanie w najnowszych koderach obrazu (stan na rok 2018). Zbiór ten zwykle zawiera: predyktor DC, predyktory kątowe, oraz predyktor PLANE.

Na dodatkowy komentarz zasługuje również kwestia przewidywania próbek bloku na podstawie odtworzonych, a nie oryginalnych próbek bloków sąsiednich. Otóż, dla każdego z bloków obrazu koder dokonuje przewidywania ich treści i przesyła do dekodera informację o tym, w jakim zakresie treści bloku nie udało się poprawnie przewidzieć, czyli jaki jest dokładnie błąd predykcji próbek bloku. Ta informacja jest potrzebna dekoderowi obrazu – błąd predykcji należy dodać do wyniku predykcji treści bloku, żeby dokonać właściwego jego odtworzenia (zdekodowania). Żeby zapewnić jednak pełną zgodność przebiegu procesów przewidywania treści w koderze i dekoderze obrazu, nie może być ono (predykcja) realizowane na podstawie oryginalnych próbek odniesienia. Takimi próbkami nie dysponuje dekoder obrazu, w przypadku stratnej jego kompresji. Sytuacja, w której koder stosuje jako odniesienie oryginalną wersję próbek (proszę zauważyć, że koder ma do nich dostęp), a dekoder do tego samego celu używa próbek o innych już wartościach (bo są wynikiem stratnej kompresji i dekompresji) prowadziłaby do **dryfu kodowania**. Pomiędzy stroną kodującą i dekodującą pojawiłaby się zatem rozbieżność rezultatów predykcji, która z uwagi na predykcyjne kodowanie danych szybko by się pogłębiała z bloku na blok obrazu (dlatego mówimy o dryfie).

W związku z powyższym faktem, koder musi się „zniżyć” do poziomu dekodera i dokonywać predykcji próbek w oparciu o wersję próbek odniesienia, którą będzie dysponował dekoder obrazu. Stąd właśnie obecność na schemacie blokowym kodera **pętli sprzężenia zwrotnego**, w której następuje wyznaczenie (dekompresja) wersji próbek, do których dostęp

będzie miał również dekoder. Pętla ta obejmuje odwrotną kwantyzację (przywrócenie skali współczynnikom DCT), odwrotne przekształcenie kosinusowe oraz dodanie do otrzymanego w ten sposób zdekodowanego błędu predykcji treści bloku wyniku jego przewidywania. Współczesny koder obrazu zawiera więc w sobie część bloków funkcjonalnych dekodera, co wpływa w pewnym stopniu na jego złożoność.

4.10.2.2. Kodowanie transformatowe błędu przewidywania treści bloków

Najnowsze kodery potrafią bardzo wydajnie przewidzieć treść kolejnych fragmentów obrazu. Pomimo tego, przewidywanie próbek nie zawsze jest idealne, w pełni trafione, i jest z nim związana pewna pomyłka, czyli błąd predykcji. Błąd ten należy wysłać do dekodera obrazu.

Z uwagi na charakter sygnału błędu predykcji próbek (sąsiednie próbki błędu są do siebie bardzo podobne – z uwagi na wysoką trafność przewidywania treści błąd ten jest najczęściej albo zerowy, albo przyjmuje bardzo małe wartości) daje się go wydajnie reprezentować przy pomocy sumy składowych harmonicznych, czyli w **dziedzinie częstotliwości**. Sygnał ten jest więc kodowany z użyciem **techniki kodowania transformatowego** (z kwantowaniem próbek widma sygnału) w sposób analogiczny do tego, który starsze kodery obrazu stosują dla próbek obrazu. Opis tego sposobu można znaleźć we wcześniejszych punktach tej książki.

Jednak w sposób analogiczny, to nie znaczy, że dokładnie tak samo. Oprócz tego, że kodowaniu podlega błąd predykcji próbek, a nie bezpośrednio próbki obrazu, to obserwuje się ponadto następujące ulepszenia:

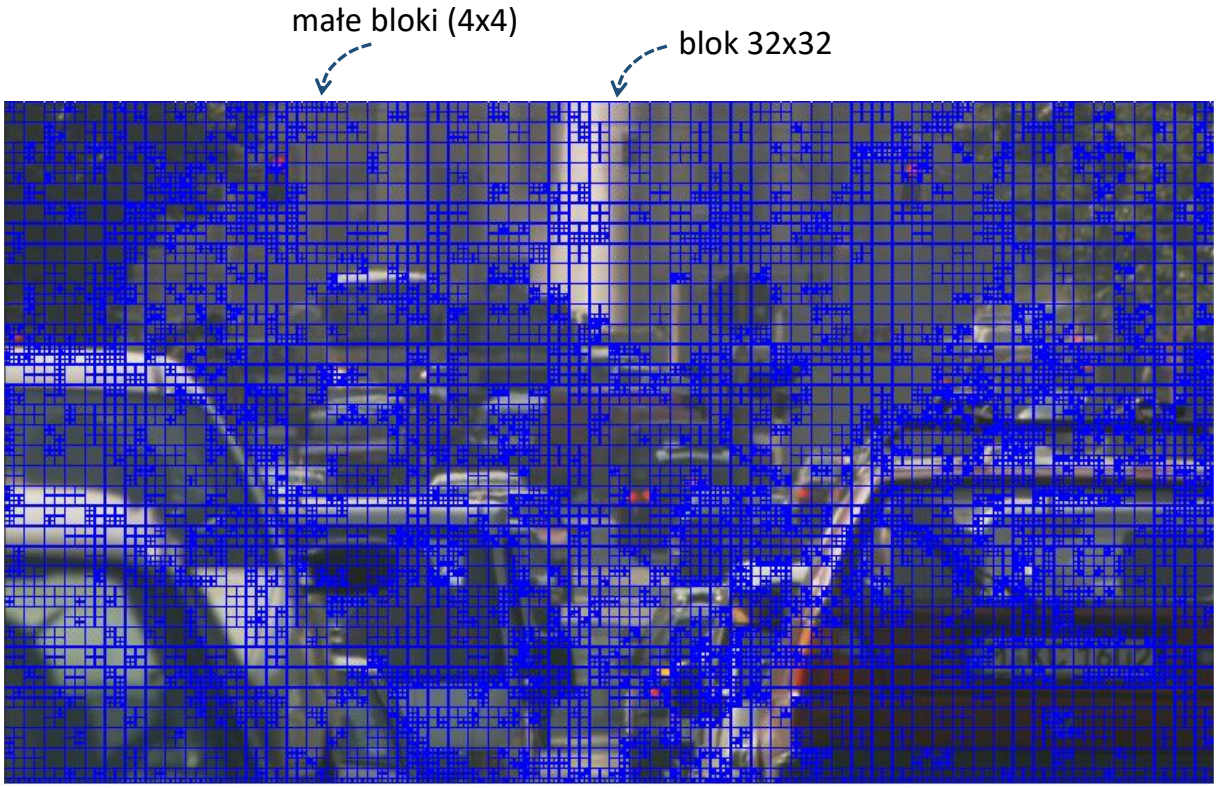
1. Za pomocą funkcji harmonicznych reprezentuje się bloki próbek błędu, których rozmiar (bloków) nie jest taki sam. Rozmiary kolejnych bloków mogą być różne (np. mogą to być rozmiary 4x4, 8x8, 16x16, 32x32) i są one wynikiem decyzji kodera obrazu. W jej podjęciu koder kieruje się oczywiście chęcią uzyskania najlepszych rezultatów kompresji obrazu. Przykładowy rezultat decyzji kodera (kodera HEVC) został przedstawiony na rysunku 4-32.

2. W starszych koderach obrazu kodowanie każdego z bloków bazowało na **DCT**. W przypadku najnowszych koderów, oprócz **DCT** stosuje się również przekształcenie **DST**, czyli takie, które opisuje sygnał za pomocą sinusów, a nie kosinusów. Przekształcenie DST wydajniej niż DCT opisuje bloki wartości, które nie wykazują aż tak silnego podobieństwa. Z tego powodu jest ono stosowane zwykle dla najmniejszych bloków, np. 4x4, i jak pokazują badania prowadzi to w takim przypadku do około 1% redukcji wielkości strumienia zakodowanych danych. Tutaj oczywiście również decyzję o sposobie kodowania podejmuje koder.

3. Część bloków próbek błędu przewidywania reprezentuje się z pominięciem transformacji, ale nadal stosując kwantowanie (mowa o tzw. trybie ‘transform skip’). Takie kodowanie nie nazywa się już oczywiście transformatowym. Najnowsze kodery ograniczają jednak stosowanie takiego kodowania do najmniejszych bloków, np. 4x4. Tryb ten pozwala na wydajniejsze kodowanie niektórych rodzajów treści, np. grafiki komputerowej czy treści z naniesionym tekstem.

Dla otrzymanych próbek widma sygnału (w przypadku stosowania przekształceń DCT lub DST) ustala się jeszcze dokładność ich reprezentacji w strumieniu danych poprzez właściwe

kwantowanie amplitud składowych (kosinusów bądź sinusów). Idea kwantowania składowych jest taka sama jak w starszych koderach.



Rysunek 4-32. Obraz testowy „Rush Hour” oraz rezultat podziału tego obrazu na **bloki transformaty** o określonym rozmiarze. Otrzymana siatka bloków jest wynikiem podjętych przez modelowe oprogramowanie HM 10.0 kodera HEVC decyzji o sposobie realizacji kodowania transformatowego błędu predykcji treści. Przykład został opracowany z użyciem oprogramowania analizatora strumienia HEVC, którego głównym wykonawcą jest Pan Piesik Daniel.

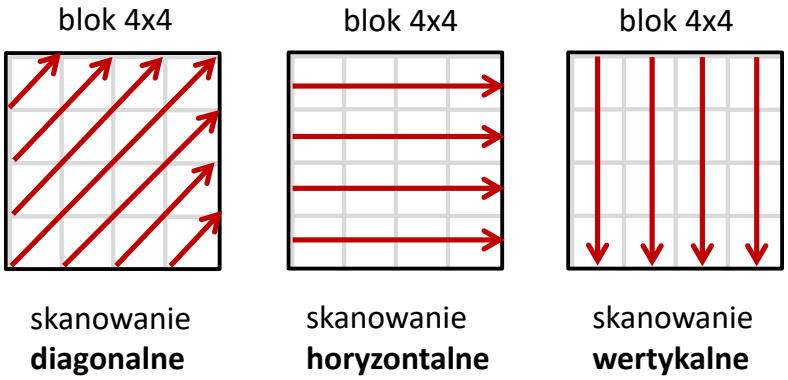
4.10.2.3. Kodowanie skwantowanych próbek widma

Skwantowane próbki widma w najnowszych koderach mają inną postać niż w koderach starszych. W związku z tym kodowanie tych danych przebiega nieco inaczej. Pierwsza zmiana to przyjęcie innego niż „zig-zag” sposobu skanowania skwantowanych współczynników transformaty. W tym miejscu zastosowanie mają trzy różne schematy porządkowania współczynników, które zostały przedstawione na poniższym rysunku. Wybór właściwego sposobu skanowania danych jest uzależniony od użytego wcześniej trybu predykcji wartości próbek obrazu. Można powiedzieć, że zasada tego wyboru jest następująca. Skanowanie wertykalne stosuje się wtedy, kiedy próbki obrazu były przewidywane z użyciem kierunku predykcji, który jest bliski horyzontalnemu. I odwrotnie, skanowanie horyzontalne w przypadku użycia kierunku predykcji bliskiego wertykalnemu. W przypadku pozostałych kierunków predykcji zastosowanie ma skanowanie diagonalne.

Bez względu na zastosowany przez koder rozmiar bloku danych, w którym stosuje się kodowanie transformatowe (blok ten będziemy nazywać blokiem transformaty) skanowanie współczynników realizuje się niezależnie w blokach 4x4. Samo kodowanie uporządkowanych już współczynników jest również inne niż w starszych koderach. Tutaj koduje się bezpośrednio pozycję ostatniego niezerowego współczynnika w bloku transformaty. Następnie, dla poszczególnych

bloków 4x4 (w przypadku bloku transformaty o rozmiarze 4x4 będziemy mieć oczywiście jeden taki blok) koduje się informację o tym, czy dany współczynnik transformaty jest faktycznie niezerowy. Jest to robione niejako w dwóch krokach. W pierwszym kroku sygnalizuje się, czy dany blok 4x4 zawiera choć jeden niezerowy współczynnik. Jeśli tak jest, i to jest krok drugi, to przesyła się następnie informację o tym, które konkretnie współczynniki w bloku są niezerowe.

Proszę zauważyć, że przyjęty, nowy sposób kodowania danych uwzględnia inny ich charakter (niż w starszych koderach), ale również użycie przez koder obrazu bloków transformaty o większym niż wcześniej rozmiarze.



Rysunek 4-33. Schematy skanowania próbek widma, które są stosowane w najnowszych koderach obrazu.

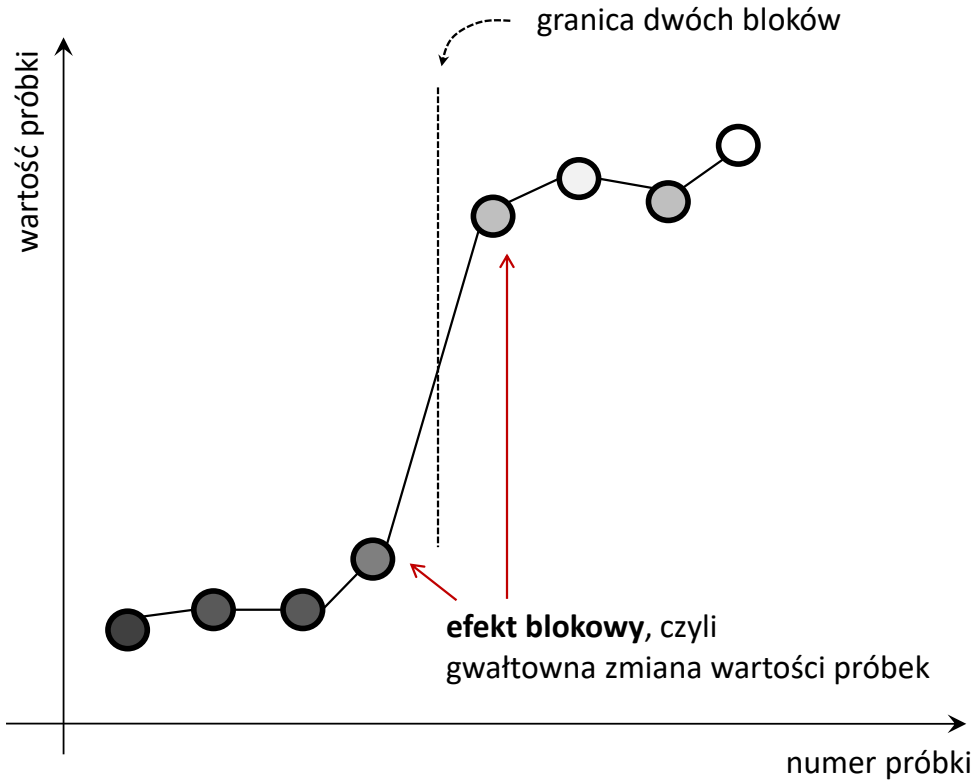
4.10.2.4. Filtracja obrazu poprawiająca jakość

Stratna kompresja obrazu prowadzi do silnej redukcji wielkości strumienia bitowego, który opisuje obraz, jednak powoduje również wystąpienie wielu przekłamań treści (zniekształceń), co pogarsza oczywiście jakość obrazu. Spośród wielu rodzajów zniekształceń treści, które wprowadza do obrazu koder transformatowy (podczas kompresji stratnej) najbardziej charakterystyczne są dwa: tzw. **efekt blokowy**, oraz przekłamania treści nazywane **efektem dzwonienia** (nazywane również **efektem tętnień**), które są związane ze znanym w teorii sygnałów **efektem Gibbsa**. Poziom wspomnianych zniekształceń jest w zakodowanym obrazie tym większy im silniej koder kwantuje próbki widma, które opisują treść obrazu.

Pomimo negatywnego wpływu obecności tych zniekształceń na postrzeganą przez widza jakość obrazu zakodowanego starsze kodery obrazu, np. JPEG, czy MPEG-2, nie podejmowały w tej kwestii żadnych działań naprawczych. Zniekształcenia te były po prostu obecne w zakodowanym obrazie, obniżając jego jakość subiektywną, i tym samym, psując efektywność kompresji danych obrazowych. Inaczej sprawa się ma w przypadku nowszych, współczesnych koderów obrazu, np. AVC, czy HEVC. Nowsze kodery dokonują próby zredukowania (choćby częściowego) błędów kompresji poprzez zastosowanie w koderze i dekoderze obrazu filtracji próbek odtworzonego (zdekodowanego) obrazu, która poprawia jego jakość. Poniższe dwa podpunkty w bardziej szczegółowy sposób omawiają temat wspomnianych zniekształceń obrazu oraz sposobów ich redukcji.

4.10.2.4.1. Efekt blokowy i sposób jego redukcji

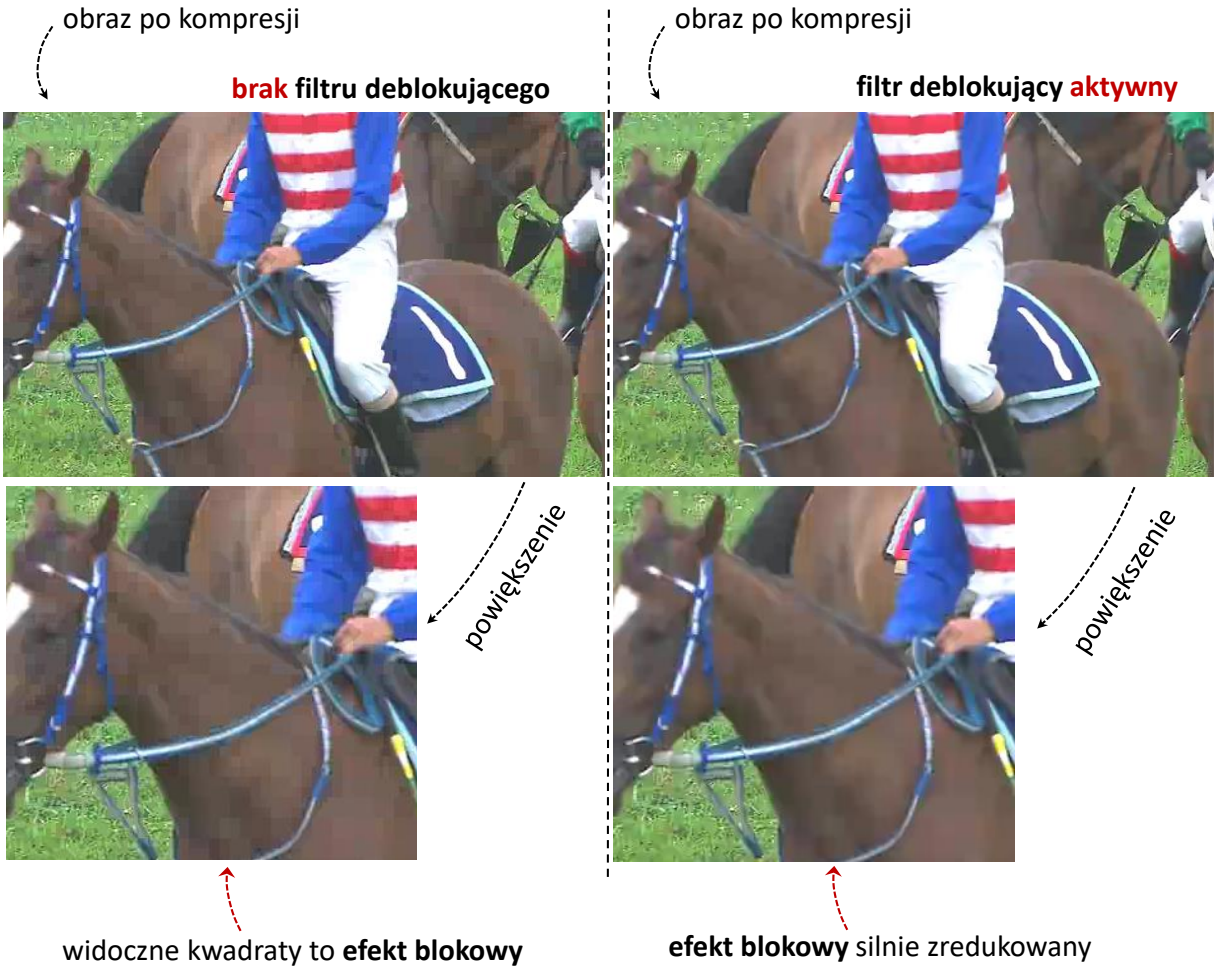
Realizowane w kolejnych blokach obrazu stratne kodowania transformatowe może doprowadzić w zdekodowanym obrazie do sytuacji, w której na granicy dwóch sąsiednich jego bloków wartości próbek zmieniają się w sposób gwałtowny. Taka sytuacja została przedstawiona na poniższym rysunku.



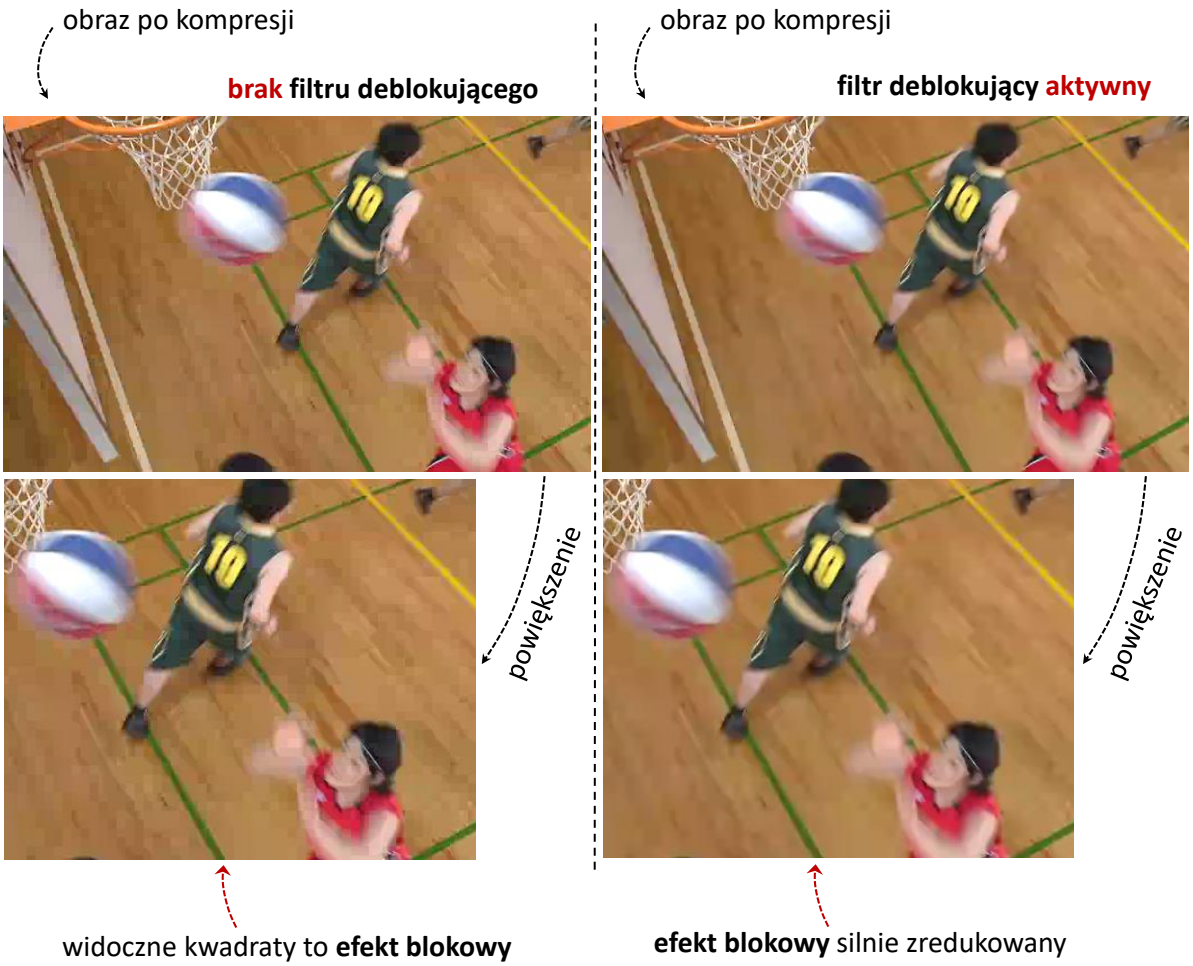
Rysunek 4-34. Ilustracja **efektu blokowego**, czyli problemu skokowej zmiany wartości próbek na granicy bloków. Efekt ten jest wynikiem stratnej kompresji obrazu.

Ta wyraźnie widoczna dla oka widza skokowa zmiana wartości próbek jest powszechnie nazywana **efektem blokowym**. W wyniku jego wystąpienia na granicy dwóch sąsiednich bloków pojawia się w obrazie pozorna krawędź (odzwierciedla ona skokową zmianę wartości próbek), która nie jest elementem treści oryginalnego obrazu, i jest w związku z tym zniekształceniem.

Źródłem tego zniekształcenia jest silna kwantyzacja próbek widma sygnału, którą koder obrazu wykonuje niezależnie w kolejnych jego blokach. Po tak przeprowadzonej kompresji treści, wartości próbek bloków bezpośrednio ze sobą sąsiadujących mogą się między sobą różnić naprawdę znacznie. W zdekodowanym obrazie objawia się to pojawieniem widocznych kwadratów (patrz rysunki 4-35 i 4-36).



Rysunek 4-35. Ilustracja rezultatu działania **filtracji deblokującej** na obrazie po stratnej kompresji. Przykłady dla obrazu sekwencji testowej „RaceHorses”. Obrazy zostały przygotowane z użyciem modelowego oprogramowania JM 18.5 kodera wizyjnego MPEG-4 AVC/H.264 [AVCSoft].

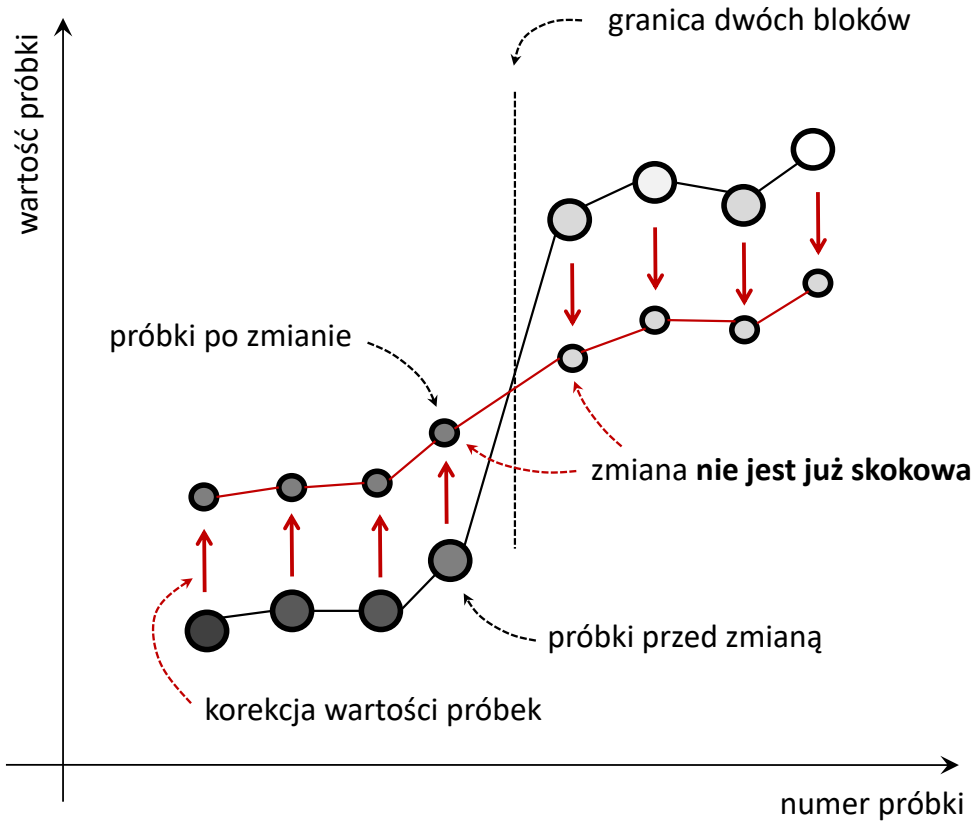


Rysunek 4-36. Ilustracja rezultatu działania **filtracji deblokującej** na obrazie po stratnej kompresji. Przykłady dla obrazu sekwencji testowej „BasketballDrill”. Obrazy zostały przygotowane z użyciem modelowego oprogramowania JM 18.5 kodera wizyjnego MPEG-4 AVC/H.264 [AVCSoft].

Jak więc widać, efekt blokowy jest bardzo dokuczliwy. W sposób istotny obniża on jakość subiektywną zdekodowanego obrazu. Dodatkowo jeszcze, obecność w obrazie opisywanego zniekształcenia utrudnia przewidywanie treści kodowanych bloków, właśnie na skutek owej różnicy wartości próbek kodowanych (oryginalnych), w stosunku do próbek bloków sąsiednich (po stratnej kompresji i rekonstrukcji). Błąd przewidywania próbek bloku przyjmuje przez to większe wartości, co prowadzi do obniżenia efektywności kompresji obrazu. Doświadczenie pokazuje, że wprowadzenie do kodera mechanizmu, który redukuje efekt blokowy poprawia efektywność kompresji o 4-5%. Dokładny wynik silnie zależy od treści kodowanych obrazów oraz stopnia nasilenia samego efektu blokowego, dlatego są przypadki, kiedy ten zysk jest jeszcze większy.

Sposobem na ograniczenie skutków efektu blokowego jest właściwa modyfikacja wartości niektórych próbek w poszczególnych blokach obrazu (obrazu po rekonstrukcji). Z tej perspektywy kluczowe jest dokonanie zmiany wartości tych próbek, które znajdują się blisko miejsca skokowej zmiany wartości. Czyli w praktyce są to próbki położone blisko granicy poszczególnych bloków. W zależności od wielkości zniekształcenia należy dodać do wspomnianych próbek pewną wartość, dodatnią bądź ujemną, żeby złagodzić efekt gwałtownego skoku wartości. Zmiany wartości próbek

dokonuje filtr, który jest określony mianem **filtru deblokującego**. Ideę tej filtracji przedstawia rysunek poniższy.



Rysunek 4-37. Ilustracja **efektu blokowego**, wraz ze sposobem jego redukcji. Kluczem do sukcesu jest właściwa modyfikacja wartości niektórych próbek.

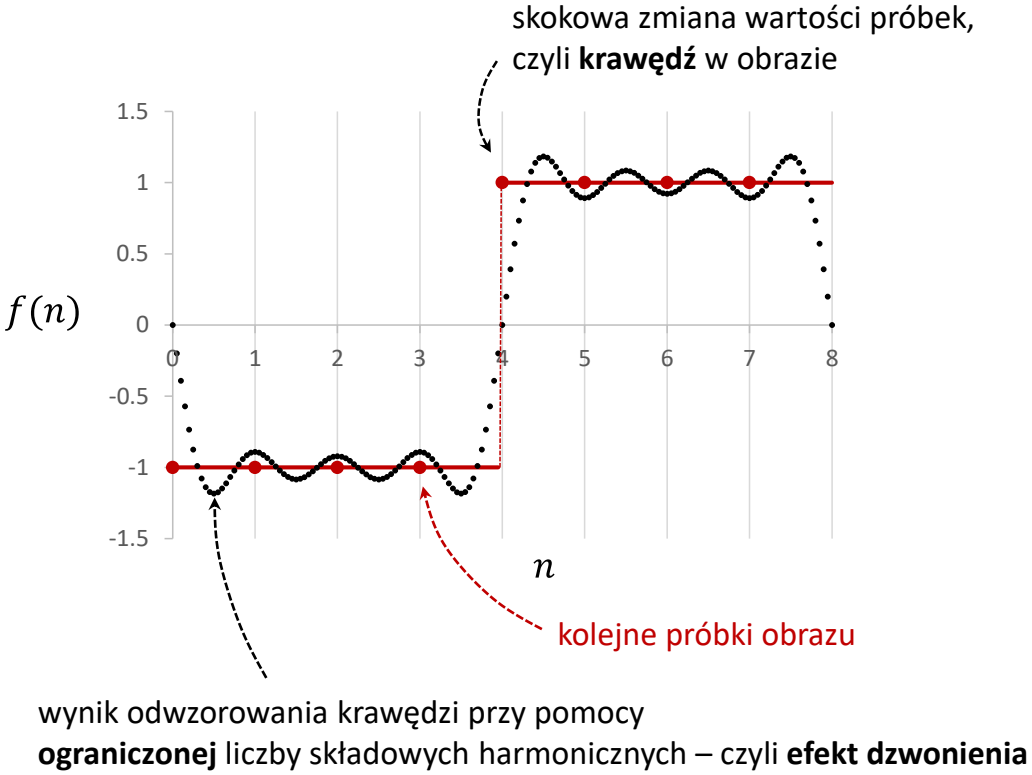
Sam sposób redukcji efektu blokowego wydaje się więc być bardzo prosty. W praktyce jednak tak nie jest. Zdekodowany obraz, oprócz krawędzi, które są bezpośrednio wynikiem efektu blokowego, zawiera również te „pożyteczne”, które stanowią element treści oryginalnego obrazu. Tych ostatnich krawędzi nie chcielibyśmy oczywiście zepsuć w trakcie filtracji. Dodatkowo, siłę tej filtracji powinno się dobierać dla każdego fragmentu obrazu osobno, z uwzględnieniem faktycznego stopnia efektu blokowego. Czyli filtracja powinna być adaptacyjna. Z wymienionych więc powodów właściwą filtrację obrazu poprzedza analiza szeregu czynników, które odnoszą się do danych obrazu oraz niektórych wyników kodowania bloków. Są to np. dane na temat trybu kodowania bloków (INTRA lub INTER – patrz również dalsza część książki), informacja o zastosowanej przez koder sile kwantowania danych w blokach, wiedza o wartościach skwantowanych próbek widma (ile współczynników jest zerowych, a ile niezerowych[43]), czy dane, które opisują ruch w sekwencji. Uwzględnienie w procesie filtracji etapu analizy danych prowadzi do rozwiązania całkiem skomplikowanego i obliczeniowo dość złożonego (około 15% złożoności całego dekodera wizyjnego, takiego jak HEVC). Jednak nagrodą jest obraz o znacznie lepszej jakości.

[43] Sposób kwantowania danych w blokach, oraz dane będące pochodną tego sposobu (np. liczba niezerowych próbek widma w blokach) są tutaj szczególnie istotne. Skoro zmiana wartości próbek jest skokowa a w blokach przeprowadzono silne kwantowanie próbek widma, to jest mocno prawdopodobne, że rozważana krawędź jest właśnie wynikiem efektu blokowego.

4.10.2.4.2. Efekt dzwonienia i sposób jego redukcji

Oprócz efektu blokowego, kwantowanie danych prowadzi także do innego, również dokuczliwego zniekształcenia. Chodzi o tzw. **efekt dzwonienia**, nazywany również **efektem tętnień** na krawędziach. Ten rodzaj zniekształcenia dotyka krawędzi obiektów obrazu, i to bez względu na ich umiejscowienie wewnątrz bloków (centrum bloku, czy brzegi bloku), i jest bezpośrednio związany ze znanym w literaturze **efektem Gibbsa**. Żeby zrozumieć przyczynę powstawania tego zniekształcenia, należy się bliżej przyjrzeć zawartości częstotliwościowej fragmentów obrazu, w których występują krawędzie.

Krawędź w obrazie odwzorowuje nagłą, skokową zmianę wartości próbek (jak na rysunku 4-38). W związku z tym, w reprezentacji częstotliwościowej takiej treści zawsze występują składowe harmoniczne o wysokich częstotliwościach (patrz Rozdział 2, w którym zawarto szczegółowe rozważania na ten temat). Mówiąc inaczej, same niskoczęstotliwościowe składowe nie byłyby w stanie tej gwałtownej zmiany wartości sygnału odwzorować. I w ten sposób dochodzimy do sedna problemu. W drodze kwantowania próbek widma bloku składowe kosinusoidalne o wysokich częstotliwościach są całkowicie usuwane (kwantowane do zera), bądź reprezentowane z bardzo małą dokładnością. Kwantowanie silnie więc redukuje składowe sygnału, które są konieczne dla poprawnego opisania skokowych zmian wartości sygnału, czyli krawędzi. Stąd właśnie wystąpienie widocznych błędów na krawędziach obiektów. Charakter tych błędów jest taki jak na zamieszczonym poniżej rysunku.



Rysunek 4-38. Ilustracja **efektu dzwonienia**, czyli problemu błędnego odwzorowania krawędzi obrazu na skutek silnego kwantowania wysokoczęstotliwościowych próbek widma sygnału.

Żeby wspomniane błędy istotnie zredukować należy we właściwy sposób zmienić wartości zdekodowanych próbek obrazu, które znajdują się w bliskim sąsiedztwie danej krawędzi. Czynność tę wykonuje się po uprzednim przeprowadzeniu filtracji deblokującej (opisanej w poprzednim punkcie). Ponieważ poziom zniekształceń może być różny w poszczególnych fragmentach obrazu, to dla każdego fragmentu oblicza się niezależnie, o jakie wartości (dodatnie bądź ujemne) należy skorygować próbki obrazu po dekompresji, żeby doprowadzić do złagodzenia efektu dzwonienia. Obliczeń tych dokonuje się porównując próbki zdekodowane z wartościami próbek w oryginalnym obrazie, a wynik tych obliczeń (w postaci wartości korygujących, które są przypisane danemu blokowi obrazu) przesyła się w zakodowanym strumieniu danych do dekodera obrazu. W ten sposób dekoder obrazu może po swojej stronie również przeprowadzić taką filtrację, pomimo że nie ma dostępu do próbek oryginalnego obrazu. Jak więc widać, opisywaną filtrację należy uznać za adaptacyjną, z uwagi na jej zdolność do dostosowania sposobu modyfikacji wartości próbek do natężenia błędów w poszczególnych fragmentach obrazu. Efekt filtracji może być taki jak na przedstawionym rysunku 4-39 – w treści obrazu proszę zwrócić szczególną uwagę na cyfrę „3” i jej otoczenie.

Rysunek 4-39. Ilustracja rezultatu działania **filtracji redukującej efekt dzwonienia** w obrazie po kompresji. Przykłady dla obrazu sekwencji testowej „RaceHorses”. Obrazy zostały przygotowane z użyciem modelowego oprogramowania HM 16.6 kodera wizyjnego HEVC.

Przedstawiona filtracja obrazu jest elementem najnowszych (stan na rok 2018) rozwiązań w zakresie kompresji obrazu, i przykładowo w technice kompresji HEVC określana jest mianem **filtracji SAO** (ang. Sample Adaptive Offset). Podobnie jak omówiona wcześniej filtracja deblokująca, filtracja SAO również się przyczynia do poprawy jakości subiektywnej obrazu oraz efektywności kompresji danych (przeciętnie o 3-4%, chociaż należy pamiętać, że rezultat ten zależy od scenariusza kodowania (sposób kwantowania danych) i treści sekwencji). Obliczeniowo, ta

filtracja jest mniej złożona niż filtru deblokującego i stanowi około 5% złożoności całego dekodera wizyjnego (np. dekodera HEVC).

4.10.2.5. Wydajność zaawansowanych technik kompresji transformatowej

4.10.2.5.1. Efektywność kompresji

Żeby ocenić efektywność współczesnych metod kompresji stratnej przeprowadzono kodowanie obrazów testowych (obrazów statycznych) z użyciem oprogramowania **x265** kodera **HEVC**. Wybór użytego kodera był podyktowany faktem, iż stanowi on bardzo wydajną (z punktu widzenia efektywności kompresji i złożoności obliczeniowej) realizację najnowszej (rok 2018) dostępnej techniki kompresji obrazu – **HEVC** (ang. High Efficiency Video Coding).

Poniżej przedstawione zostały rezultaty kodowania, jakie uzyskano dla przykładowego obrazu testowego „PartyScene”. Zamieszczone obrazy prezentują wyniki kompresji dla kilku różnych scenariuszy kodowania – od kompresji stosunkowo słabej, z którą wiąże się wysoka jakość zakodowanego obrazu, aż po kompresję silną, która daje w wyniku obrazy o niskiej jakości. Żeby właściwie ocenić poziom zniekształceń obrazu po kompresji, warto oglądać poniższe rysunki przy odpowiednim powiększeniu, odnosząc jednocześnie subiektywną jakość obrazów do jakości obrazu oryginalnego, czyli nieskompresowanego.

Analiza otrzymanych wyników pozwala na sformułowanie następujących, generalnych wniosków. Obrazu (po kompresji i dekompresji) o bardzo dobrej jakości subiektywnej można się spodziewać, jeśli kodowanie obrazu jest zorientowane na uzyskanie współczynnika kompresji na poziomie **kilka – dwadzieścia kilka**. W tym przypadku zauważenie w obrazie zdekodowanym artefaktów kompresji jest bardzo trudne lub wręcz niemożliwe. W przypadku kompresji około **30-krotnej** zaczynają być już widoczne błędy kompresji, np. rozmycie szumu będącego treścią oryginalnego obrazu, jednak problem ten dotyczy z reguły tylko niektórych fragmentów obrazu. Inaczej jest w przypadku kompresji jeszcze silniejszej, powiedzmy około **50-krotnej**, kiedy to wyraźnie widać już artefakty kodowania obrazu w wielu jego fragmentach. Wspomniane przekłamania treści (patrz rysunek 4-44) nie mają jednak charakteru wyraźnego efektu blokowego, czy wyraźnego efektu dzwonienia, z uwagi na zastosowanie w koderze x265 odpowiednich filtracji na zrekonstruowanych próbkach, które wspomniane błędy silnie redukują. Bardzo silne zniekształcenia obrazu, a tym samym wrażenie niskiej jego jakości pojawiają się w przypadku stopnia kompresji, który przekracza **100**. Jakość obrazu jest tutaj jednak znacznie, znacznie lepsza niż w przypadku analogicznej kompresji prostym koderem, np. JPEG.

Zdaniem autora przedstawione wyniki dobrze obrazują możliwości współczesnych technik kompresji w zakresie kodowania statycznego (nieruchomego) obrazu, jednak należy mieć świadomość, iż na dokładne rezultaty duży wpływ mają treść kodowanych obrazów oraz sposób, w jaki koder wybiera tryby kompresji. Zamieszczone w tym punkcie wyniki dotyczą takiego sposobu wyboru trybów w koderze x265, który dobrze realizuje kompromis pomiędzy efektywnością kodowania danych i czasem obliczeń, które wykonuje koder. Jest to więc scenariusz, który jest w praktyce najczęściej stosowany.

Rysunek 4-40. Pierwszy obraz sekwencji „PartyScene”. Obraz **oryginalny, nieskompresowany**.

Rysunek 4-41. Zdekodowany obraz po wcześniejszej kompresji. Stopień kompresji wynosi **17,15**. Ciągle niewidoczne błędy kodowania.

Rysunek 4-42. Zdekodowany obraz po wcześniejszej kompresji. Stopień kompresji wynosi **28,73**. W niektórych fragmentach obrazu zaczynają być widoczne błędy, np. rozmycie szumu, który przed kompresją był widoczny na podłodze (uwaga: szum ten był elementem treści oryginalnego obrazu).

Rysunek 4-43. Zdekodowany obraz po wcześniejszej kompresji. Stopień kompresji wynosi **39,98**. W niektórych fragmentach obrazu zaczynają być widoczne błędy, np. rozmycie szumu, który przed kompresją był widoczny na podłodze.

Rysunek 4-44. Zdekodowany obraz po wcześniejszej kompresji. Stopień kompresji wynosi **51,76**. Dość mocno widoczne artefakty kodowania w wielu fragmentach obrazu, np. podłoga, donice, pudełka.

Rysunek 4-45. Zdekodowany obraz po wcześniejszej kompresji. Stopień kompresji wynosi **112,62**. Silne artefakty kompresji w całym obrazie.

4.10.2.5.2. Złożoność obliczeniowa

Współczesne kodeki obrazu (jak np. HEVC) stawiają niestety bardzo wysokie wymagania, jeśli chodzi o ilość obliczeń, które należy wykonać podczas kompresji czy dekompresji obrazu. W tym punkcie mowa jest tylko o złożoności kodowania statycznego, czyli nieruchomego obrazu.

I tak, zależnie od stopnia kompresji danych działający na procesorze **Intel Core i7 – 4770** (taktowanego zegarem **3,4GHz**, przy 8 aktywnych wątkach obliczeniowych) program kodera **x265** potrafi w czasie jednej sekundy zakodować od 1,5 do 3,5 obrazów o przestrzennej rozdzielczości full HD (1920x1080). Daje to czas kodowania jednego obrazu od około 290 ms do 600 ms. Takie czasy kodowania zostały otrzymane w przypadku kompresji obrazów sekwencji „BasketballDrive” i „BQTerrace” ze współczynnikiem kompresji na poziomie odpowiednio 100 oraz kilkanaście. Widać więc, że im mniejszy jest stopień kompresji danych tym dłużej trwa ich kodowanie. Raz jeszcze należy podkreślić, że przytoczone czasy dotyczą kodowania z 8 krotnym zrównolegleniem obliczeń w koderze obrazu.

Powyższe dane dotyczą konfiguracji kodera x265, w którym wykorzystuje się tylko narzędzia kompresji przewidziane dla kodowania statycznego obrazu, i to w sposób, który jest „złotym środkiem” z perspektywy złożoności kodowania i efektywności kompresji. Tak więc, zmieniając w koderze sposób wyboru trybów można wspomniane czasy dość wyraźnie zmienić – skrócić bądź wydłużyć. Przykładem jest inna (od poprzednio analizowanej) konfiguracja kodera x265, której najważniejszym priorytetem jest prędkość kodowania obrazów. Przy tej konfiguracji możliwe stało się kodowanie od 6 do 13 obrazów w ciągu sekundy na tej samej platformie testowej. Odpowiada to czasom kodowania pojedynczego obrazu (full HD) od około 80 ms do blisko 170 ms. Jednak ceną tego szybszego działania kodera jest zauważalny spadek efektywności kompresji danych. Badania autora wskazują, że zależnie od treści kodowanego obrazu spadek efektywności wynosi zwykle kilka – kilkanaście procent.

Warto podkreślić, że podane w tym punkcie czasy kodowania obrazu dotyczą programu x265 kodera obrazu, w którym realizowanych jest już szereg technik optymalizacji kodu, które przyspieszają jego działanie (np. realizowanie obliczeń z wykorzystaniem multimedialnych rozszerzeń współczesnych procesorów, poprzez użycie instrukcji SIMD, czy realizowanie obliczeń na 8 dostępnych w procesorze wątkach obliczeniowych). Złożoność dekodera obrazu, w którym zastosowane zostały podobne techniki przyspieszenia obliczeń co w koderze, jest wielokrotnie mniejsza – nawet 20-30 razy! (rezultat porównania czasów kodowania i dekodowania obrazów programami kodera **x265** i dekodera **ffmpeg**, na procesorze **Intel Core i7 – 4770**, **3,4GHz**, przy 8 aktywnych wątkach obliczeniowych). Z uwagi na użycie podczas testów szybkich implementacji kodera i dekodera obrazu można powiedzieć, że przedstawione w tym punkcie wyniki dobrze odzwierciedlają faktyczne możliwości współczesnych kodeków przemysłowych, zrealizowanych w postaci programu. Dalsze skrócenie czasów kodowania i dekodowania obrazu jest oczywiście ciągle możliwe, stosując np. większą liczbę wątków obliczeniowych, czy realizując kodek w postaci dedykowanego układu scalonego.

Część II – Kompresja ruchomego obrazu

4.11. Kodowanie ruchomego obrazu – najprostsze podejście

Koncepcyjnie najprostszym sposobem reprezentacji ruchomego obrazu jest całkowicie niezależna od siebie kompresja każdego z obrazów sekwencji. Przy takim podejściu można użyć w zasadzie którejkolwiek z metod kompresji treści, które zostały przedstawione w tej książce do tej pory. Z uwagi jednak na uzyskiwaną efektywność kompresji danych szczególnie popularnym rozwiązaniem jest połączenie mechanizmu przewidywania treści kolejnych fragmentów obrazu z transformatowym kodowaniem otrzymanych danych resztkowych. Ponieważ treść fragmentów przewiduje się w oparciu o dane, które pochodzą z tego samego obrazu (bo założyliśmy tutaj, że każdy z obrazów sekwencji kodujemy niezależnie od siebie) to takie kodowanie zostało nazwane **predykcyjnym wewnątrzobrazowym**.

Jednak z perspektywy kodowania ruchomego obrazu, użycie nawet najlepszych metod **kompresji wewnątrzobrazowej** nie zapewni zadowalających efektów reprezentacji obrazów. Można się o tym łatwo przekonać zestawiając ze sobą dwie wielkości: 1) rozmiar danych, które przedstawiają oryginalne obrazy sekwencji wizyjnej, oraz 2) efektywność technik kodowania obrazu nieruchomego (czyli statycznego).

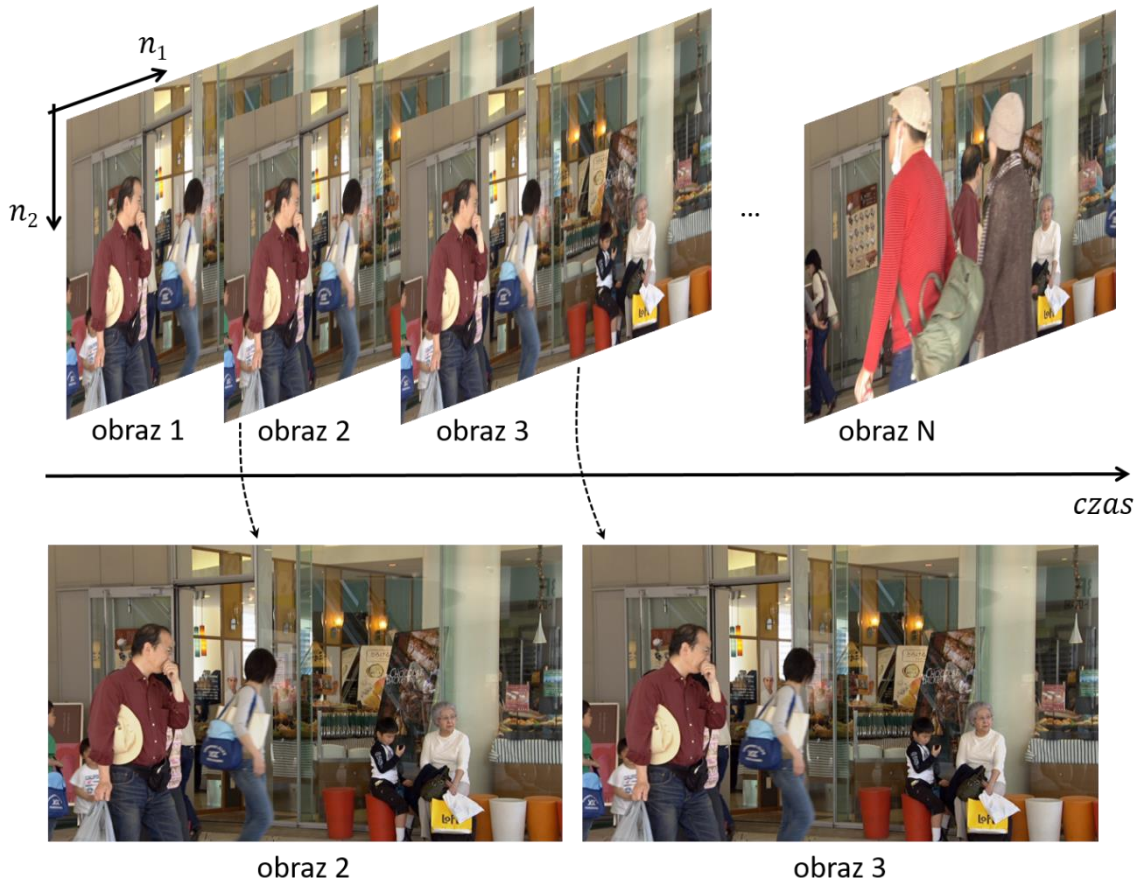
W przypadku sekwencji obrazów o wysokiej rozdzielczości przestrzennej, np. 1920x1080 punktów obrazu i rozdzielczości czasowej, np. 60 obrazów na sekundę oryginalne dane to aż blisko 3 Gb/s (gigabita na sekundę), gdyby założyć standardową reprezentację każdej próbki kolorowego obrazu na 24 bitach (8 bitów na próbkę każdej z trzech składowych kolorowego obrazu). Jest to więc ogromna wprost ilość danych. Jak pokazaliśmy w poprzednich punktach wydajność współczesnego kodera obrazu statycznego (np. kodera HEVC, który stosuje tylko i wyłącznie metody kodowania wewnątrzobrazowego) pozwoliłaby na prawie **30-krotne** zredukowanie liczby bitów, zapewniając przy tym dobrą lub bardzo dobrą jakość obrazów po kompresji. Na pierwszy rzut oka wydaje się, że 30 razy to dużo. W takim przypadku zamiast 3 Gb/s po takiej kompresji otrzymalibyśmy znacząco mniejszy strumień danych o wielkości 100 Mb/s. I tak jest w istocie. Jednak z drugiej strony podana efektywność kodowania obrazów jest ciągle niewystarczająca z perspektywy ogromnej liczby systemów z transmisją ruchomego obrazu, np. telewizji cyfrowej, telewizji w Internecie, czy chociażby systemów mobilnych. Potrzebne są zatem zupełnie nowe narzędzia kompresji obrazów, które ten ostatni strumień danych jeszcze dodatkowo znacznie zmniejszą.

4.12. Nadmiarowość czasowa sekwencji – cenna wskazówka dla jeszcze wydajniejszej kompresji

W typowym obrazie ruchomym, w czasie każdej sekundy sekwencji znajduje się kilkadziesiąt statycznych obrazów (np. 30, 50, 60 obrazów). Obrazy te przedstawiają wygląd sceny w kolejnych chwilach czasu, przy czym odstęp między tymi chwilami jest bardzo mały, bo wynosi np. 1/60 s.

Z uwagi więc na tak dużą bliskość czasową kolejnych obrazów sekwencji obrazy, które bezpośrednio ze sobą sąsiadują wykazują olbrzymie wprost podobieństwo. Jest tak również w przypadku obrazów, które prezentują szybkozmienną scenę. Ruch obiektów sceny wpływa

oczywiście na ich położenie w obrazach z kolejnych chwil czasowych, jednak znakomitą część informacji, która występuje w danym obrazie można również znaleźć w obrazach z innych chwil czasowych. Dobrze to widać na przedstawionym poniżej przykładzie, który pokazuje dwa kolejne obrazy sekwencji wizyjnej. Obrazy te są faktycznie bardzo, ale to bardzo do siebie podobne.



Rysunek 4-46. Reprezentacja ruchomej sceny przy pomocy sekwencji statycznych obrazów. Obrazy sekwencji przedstawiają widok sceny w kolejnych chwilach czasu i obrazy te wykazują bardzo silne podobieństwo.

Obrazy sekwencji wykazują więc gigantyczną **nadmiarowość czasową**. Wspomniana nadmiarowość, daje zatem możliwość bardzo wydajnego przewidywania kodowanej treści odwołując się nie do innych fragmentów tego samego obrazu, ale do tego, co już jest w innym obrazie, bądź obrazach, które już wcześniej zostały zakodowane i przesłane do dekodera. Mówimy więc tutaj o szczególnym sposobie predykcyjnego kodowania treści obrazów, bo wykorzystującym wiedzę o treści obrazów z innych chwil czasu. Z tego więc powodu kodowanie to jest określane mianem **kodowania z międzyobrazową predykcją treści**, a obrazy na podstawie których ta treść jest przewidywana **obrazami odniesienia**.

Jednak podobnie jak miało to miejsce w przypadku wewnątrzobrazowej predykcji treści, również i tutaj przewidywanie danych obrazu nie zawsze będzie idealne, bezbłędne. W ogólności zostanie popełniony pewien błąd, który tak jak wcześniej, jest nazywany **błędem predykcji** treści. Z uwagi jednak na bardzo wysoką skuteczność przewidywania treści wynikowy błąd predykcji próbek będzie można opisać przy pomocy dużo mniejszej liczby bitów niż gdybyśmy bezpośrednio kodowali te próbki lub nawet przewidywali je w drodze wewnątrzobrazowej predykcji. Błąd

predykcji próbek będzie więc w koderze przedmiotem dalszej kompresji. Kodowanie tego sygnału przebiega w praktyce dokładnie tak samo (co do zasadniczej idei) jak sygnału błędu, który wcześniej był wynikiem predykcji wewnątrzobrazowej. Czyli zastosowanie mają tutaj metody kodowania transformatowego danych z entropijną kompresją końcowych danych.

Jednak na ten moment, najważniejsze pytanie, to jak wydajnie przewidywać treść obrazu wykorzystując w tym celu obrazy z innych chwil czasowych? Jak to wydajnie robić w przypadku ruchomej, czyli zmieniającej się w czasie sceny?

4.13. Jak wydajnie przewidywać obrazy sceny, która zmienia się w czasie?

W przypadku omówionej wcześniej wewnątrzobrazowej predykcji treści nie przewidywało się wszystkich próbek obrazu za jednym zamachem, stosując dla każdej próbki dokładnie ten sam schemat przewidywania jej wartości. Stwierdzono, że takie podejście byłoby nieefektywne. Zamiast tego dokonywano podziału obrazu na mniejsze fragmenty, bloki o określonym rozmiarze, i dla każdego z nich indywidualnie wybierano najlepszy sposób predykcji ich treści. Algorytm przewidywania traktował więc odrębnie każdy z fragmentów obrazu, uwzględniając specyfikę zawartej w nim treści.

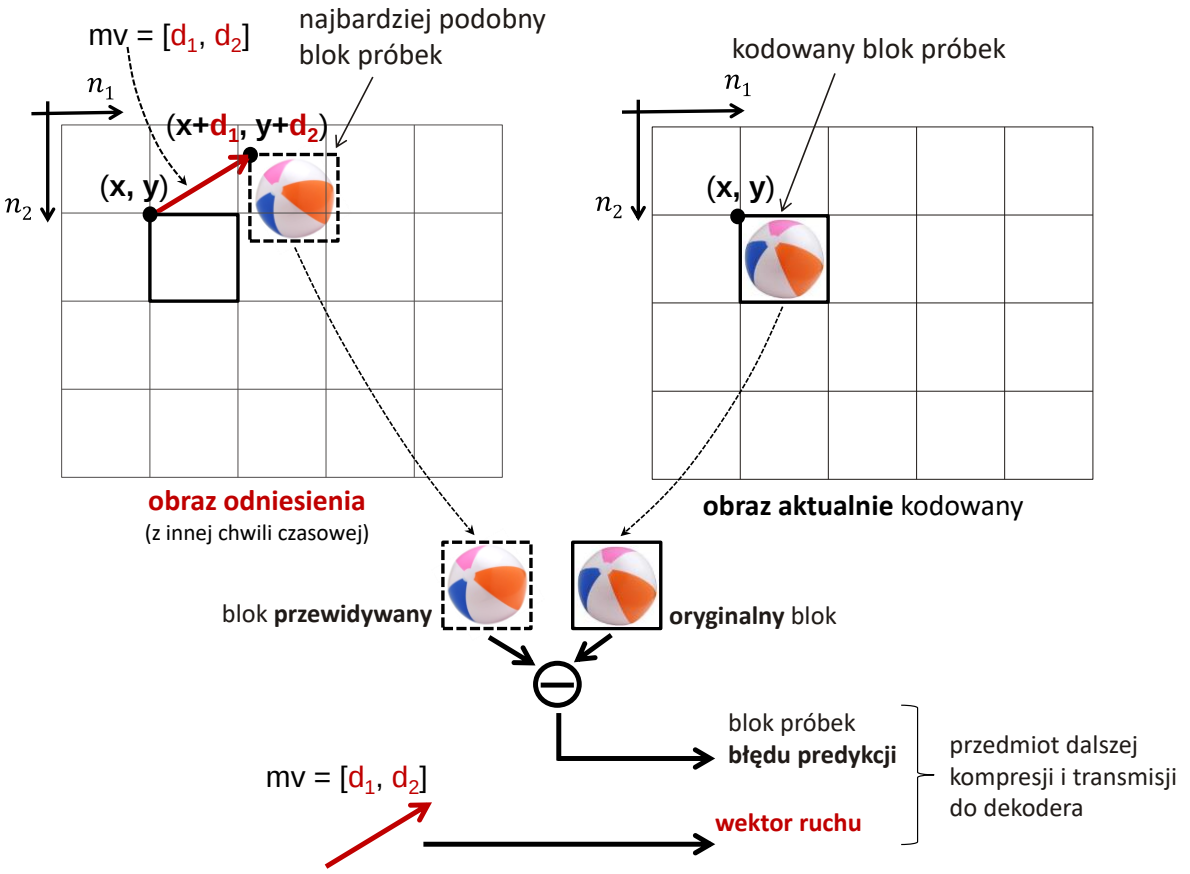
Co do głównej zasady dokładnie tak samo jest w przypadku przewidywania międzyobrazowego. Predykcję przeprowadza się niezależnie w wyznaczonych wcześniej fragmentach obrazu. W stosunku do kodowania wewnątrzobrazowego w kodowaniu międzyobrazowym inne jest tylko pochodzenie danych, które stanowią odniesienie dla wydajnego przewidzenia wartości próbek bloku, który aktualnie się przetwarza. Tak jak powiedziano, dane te pochodzą z innego obrazu.

W przypadku omawianego w tym punkcie sposobu przewidywania próbek największym niewątpliwie problemem jest ruch obiektów sceny, jak również zachodzące między obrazami inne typy zmiany ich treści. Wszystkie te zmiany znakomicie utrudniają przewidywanie danych. Trafne wychwycenie zmian treści jest więc kluczem do sukcesu, jakim jest wydajna predykcja fragmentów obrazu. W praktyce zadanie to może być bardzo trudne i szalenie czasochłonne. Obiekty sceny mogą zmieniać swoje położenie w obrazach, podlegając chociażby prostemu ruchowi translacyjnemu na płaszczyźnie obrazu (ruch translacyjny, czyli obiekty mogą się poruszać w lewo, w prawo, góra, dół). Ale treść fragmentów obrazu może się zmieniać również w inny, bardziej złożony sposób, np. zgodnie z operacjami powiększenia, pomniejszenia, obrotu o pewien kąt (wyznaczanego względem jakiegoś punktu), czy w oparciu o inne, jeszcze trudniejsze przekształcenia. Z praktycznego punktu widzenia nie jest więc w ogólności możliwe wychwycenie wszystkich tych typów zmian, poświęcając na to umiarkowane nakłady obliczeniowe.

Dlatego z tego powodu, starając się utrzymać złożoność przewidywania treści w jakichś rozsądnych ramach, przyjmuje się zwykle założenie, że dokonujące się między obrazami zmiany treści są wynikiem samych tylko prostych przesunięć bloków w dowolnym kierunku (lewo, prawo, góra, dół). W koderach obrazu powszechnie przyjmuje się zatem bardzo uproszczony **model ruchu translacyjnego** treści, która jest zawarta w blokach.

Kodując więc dany blok obrazu (zgodnie z przyjętym sposobem predykcji treści) koder zadaje sobie następujące pytanie: czy aktualnie kodowana treść lub treść do niej podobna pojawiła się już w którymś z wcześniej zakodowanych obrazów? Jeśli tak, to jakie jest jej (tej kodowanej treści) położenie w obrazie, który stanowi odniesienie dla predykcji próbek kodowanego bloku? Z uwagi na wspomniane silne podobieństwo kolejnych obrazów jest wysoce prawdopodobne, że

koder faktycznie znajdzie w obrazie odniesienia blok z taką treścią. Biorąc pod uwagę ruch obiektów sceny inne może być jego położenie wewnątrz obrazu. I to położenie należy jednoznacznie wskazać. Zadaniem kodera jest więc obliczenie składowych d_1 i d_2 **wektora**, który określi względne (ponieważ względem punktu o współrzędnych (x, y) – patrz rysunek poniżej) położenie w obrazie odniesienia najbardziej podobnego bloku do tego, który aktualnie kodujemy. Składowa d_1 wektora jest składową poziomą i określa wielkość przesunięcia bloku próbek w poziomie (przesunięcia, które się dokonało między obrazami: odniesienia i tym aktualnie kodowanym). Składowa pionowa d_2, wyznacza wielkość przesunięcia w kierunku pionowym. W przyjętym modelu ruchu obiektów sceny wektor ten ma odzwierciedlać (przynajmniej tak jest w teorii) faktyczny kierunek ruchu obiektów w czasie, dlatego powszechnie nazywa się go **wektorem ruchu**. Powtarzając tę czynność dla wszystkich bloków kodowanego obrazu otrzymujemy więc swego rodzaju mapę (jest nią właśnie wyznaczony zbiór wektorów ruchu), która odzwierciedla kierunki zmian treści poszczególnych bloków, jaka dokonała się między obrazami. Z uwagi na kojarzenie wektorów ruchu z ruchem obiektów sceny proces ich wyznaczania w koderze jest nazywany **estymacją ruchu**. Ponieważ realizowana predykcja bloku próbek korzysta z obliczonych wektorów ruchu, które dają możliwość skompensowania wpływu ruchu obiektów sceny na treść obrazu, to jest to **predykcja z estymacją i kompensacją ruchu**. Jeśli do predykcji używa się obrazu odniesienia, który na osi czasu znajduje się przed obrazem kodowanym, czyli wcześniej od niego, to znaczy, że jest to **predykcja wprzód**, bo przewidujemy treść obrazu do przodu. Dodatkowo, przewidywanie ma miejsce z jednego kierunku w czasie, więc jest to **predykcja jednokierunkowa**.



Rysunek 4-47. Jednokierunkowa międzyobrazowa predykcja treści obrazu. Aktualnie kodowany obraz jest przewidywany w oparciu o treść jednego obrazu odniesienia.

4.14. Estymacja ruchu w ujęciu matematycznym

Z matematycznego punktu widzenia estymacja ruchu jest więc zagadnieniem optymalizacji, w którym dla zadanego problemu poszukuje się najlepszego rozwiązania. Tym rozwiązaniem jest określony blok próbek z obrazu odniesienia, który wykazuje największe podobieństwo z blokiem aktualnie kodowanym.

W przypadku sposobu estymacji ruchu, który został przedstawiony w poprzednim punkcie algorytm optymalizacji poszukuje więc takich wartości składowych wektora ruchu, które minimalizują wskaźnik $\boldsymbol{D}$ wyrażający różnicę między blokiem aktualnie kodowanym i blokiem próbek, które pochodzą z obrazu odniesienia. Wskaźnik ten można zdefiniować na wiele różnych sposobów, np., patrz poniższy wzór, jako sumę modułów różnic odpowiadających sobie próbek bloku kodowanego i rozważanego bloku z obrazu odniesienia (którego względne położenie w obrazie odniesienia określa właśnie wektor ruchu) lub jako sumę kwadratów różnic tych próbek, lub jeszcze inaczej. Wskaźnik $\boldsymbol{D}$ jest więc funkcją celu wspomnianej optymalizacji, funkcją, która zawiera dwie zmienne – są nimi składowe $\boldsymbol{d_1}$ i $\boldsymbol{d_2}$ wektora ruchu. W rozważanym przypadku estymacja ruchu jest więc problemem optymalizacji dwuwymiarowej funkcji celu. Zważywszy na sposób działania algorytmu optymalizującego (poszukiwanie w obrazie odniesienia bloku próbek, który względem przyjętego kryterium podobieństwa dwóch bloków jest najbardziej podobny do aktualnie kodowanego bloku próbek) przedstawiony sposób estymacji ruchu znany jest jako metoda **pasowania bloków**. Ten sposób estymacji ruchu znajduje powszechne praktyczne zastosowanie w koderach wizyjnych [Jain81].

$$D(x,y;\boldsymbol{d_1},\boldsymbol{d_2}) = \sum_{l_1=0}^{N_1-1}\sum_{l_2=0}^{N_2-1}\left|f_{blok\ kodowany}(x+l_1,y+l_2) - f_{blok\ odniesienia}(x+l_1+\boldsymbol{d_1},y+l_2+\boldsymbol{d_2})\right| \tag{4.1}$$

gdzie:

(x,y) – współrzędne (w obrazie) lewego górnego narożnika bloku próbek, dla którego poszukuje się najbardziej podobnego bloku próbek w obrazie odniesienia;

$f_{blok\ kodowany}(x+l_1,y+l_2)$ – funkcja opisująca wartości próbek aktualnie kodowanego bloku (elementy w nawiasie określają położenie próbek wewnątrz obrazu);

$f_{blok\ odniesienia}(x+l_1+\boldsymbol{d_1},y+l_2+\boldsymbol{d_2})$ – funkcja opisująca wartości próbek bloku z obrazu odniesienia (elementy w nawiasie określają położenie próbek wewnątrz obrazu);

$[\boldsymbol{d_1},\boldsymbol{d_2}]$ – składowe poszukiwanego wektora ruchu;

$\boldsymbol{N_1}, \boldsymbol{N_2}$ – rozmiar (w próbkach) bloku próbek.

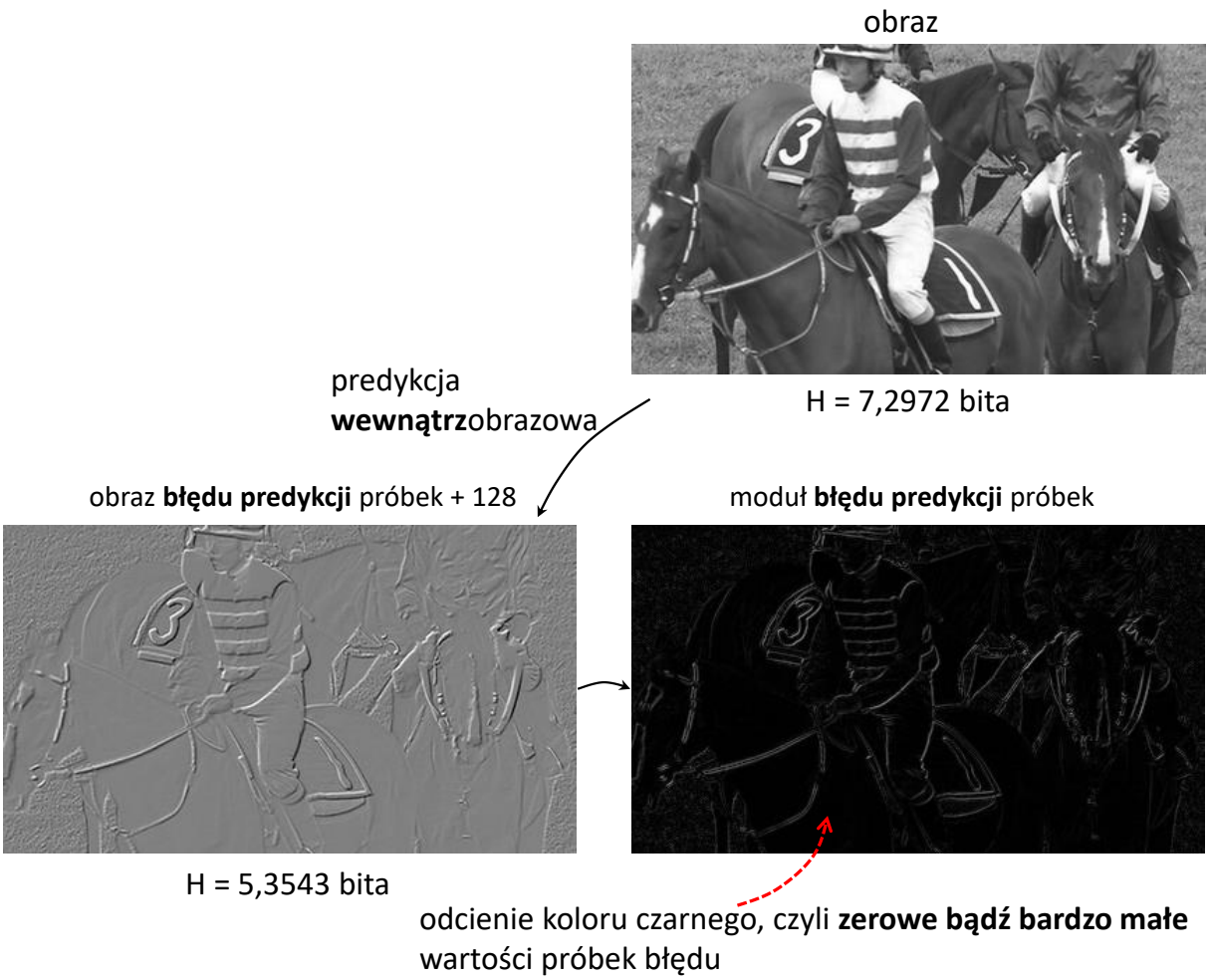
4.15. Jaka jest skuteczność międzyobrazowej, jednokierunkowej predykcji treści?

Na tak postawione pytanie można od razu odpowiedzieć jednym zdaniem – jest ona wyraźnie większa niż predykcji wewnątrzobrazowej. Łatwo to zaobserwować porównując ze sobą rzeczywiste rezultaty predykcji treści (a w zasadzie wyniki błędu przewidywania próbek obrazu wraz z wartościami entropii próbek błędu), jakie dla wybranego obrazu sekwencji „RaceHorses”

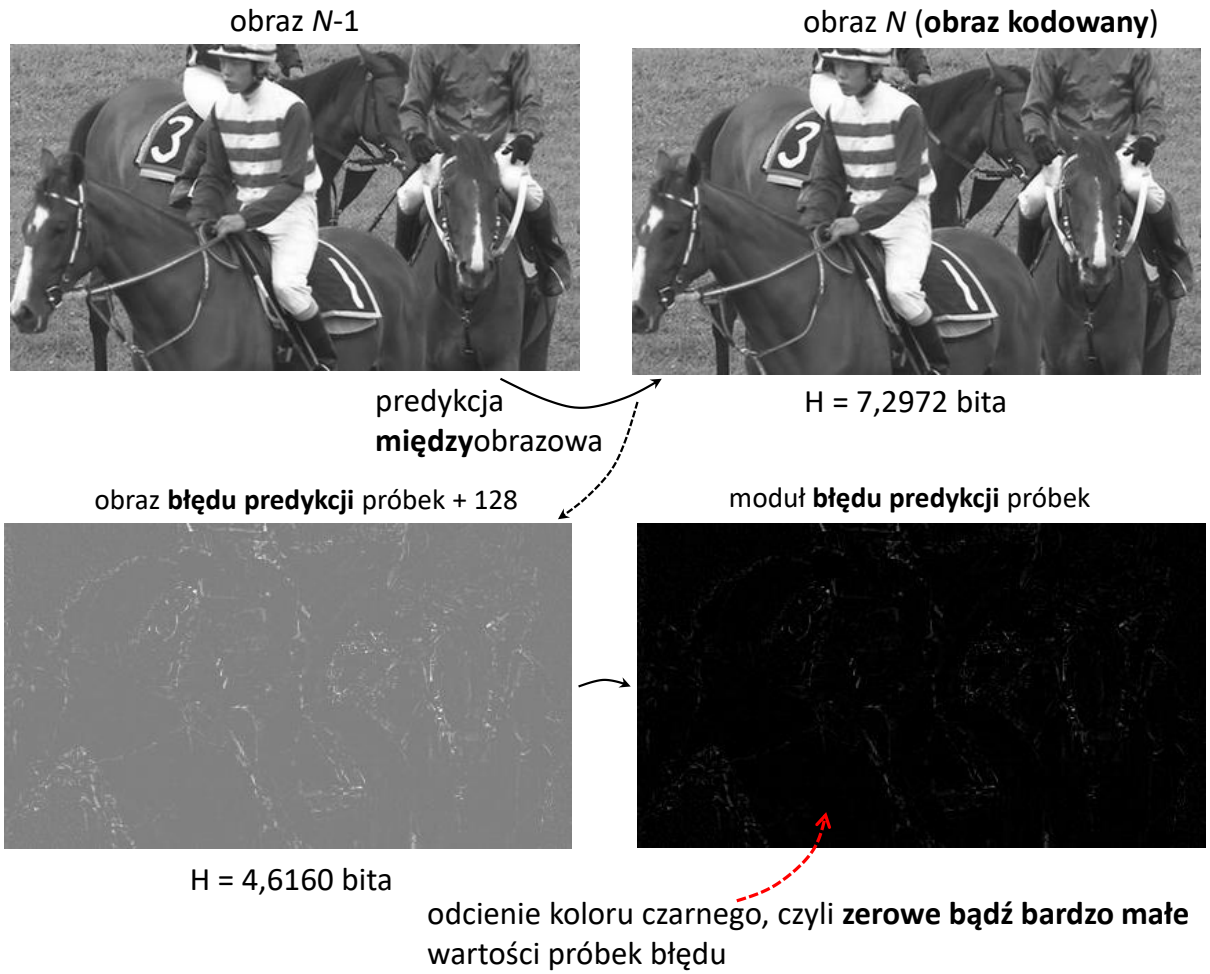
zostały otrzymane dla dwóch rodzajów predykcji: **wewnątrzobrazowej** i **międzyobrazowej** (patrz rysunki 4-48 i 4-49). Analiza tych wyników prowadzi do następujących ogólnych obserwacji.

Predykcja wewnątrzobrazowa potrafi trafnie przewidywać jedynie mało skomplikowane, jednolite fragmenty obrazu. Błąd przewidywania takich fragmentów jest relatywnie mały (i to dobrze, bo zwykle takich fragmentów jest w obrazie dużo). Ale niektóre fragmenty obrazu przedstawiają przecież złożoną treść, z dużą liczbą krawędzi i innych drobnych szczegółów. Przewidywanie takiej treści, na podstawie treści sąsiednich bloków tego samego obrazu jest ekstremalnie trudne i skutkuje w praktyce większymi wartościami próbek błędu. Widać to szczególnie dobrze w wynikach przewidywania tych części obrazu, w których zmiana wartości próbek była nagła, skokowa (patrz rysunek 4-48).

Inaczej jest w przypadku przewidywania międzyobrazowego. Tutaj, z określonej lokalizacji obrazu odniesienia (lokalizacja ta jest wskazywana przez wektor ruchu) kopiowana jest treść fragmentu obrazu, o którym mówimy, że w największym stopniu przypomina fragment, który aktualnie kodujemy (czyli jest do fragmentu kodowanego najbardziej podobny). W ten sposób, nawet w przypadku kodowania złożonych fragmentów obrazu przewidywanie ich treści może być bardzo dokładne. Oczywiście pod warunkiem, że w obrazie odniesienia udaje się taki złożony, podobny fragment faktycznie odszukać. Praktyka pokazuje, że najczęściej tak. Proszę jednak pamiętać, że kodowaniu podlegają obrazy ruchomej sceny. Mamy więc tutaj ruch obiektów, który może prowadzić do ich wzajemnego zasłaniania się. Poruszające się obiekty mogą także zasłaniać treść obrazu, która stanowi tło. Możliwe są również zmiany oświetlenia sceny, które różnicują treść kolejnych obrazów, itd. Z tych powodów nie każdą treść, która w kodowanym obrazie podlega kompresji udaje się dokładnie przewidzieć posiłkując się informacją z obrazu odniesienia. W przypadku pewnych fragmentów obrazu ich przewidywanie będzie więc obarczone większym błędem. Na szczęście, takich fragmentów obrazu jest bardzo, bardzo niewiele, na co wskazuje zamieszczony na rysunku 4-49 wynik.



Rysunek 4-48. Prezentacja skuteczności **wewnątrzobrazowej** predykcji treści. Większy błąd przewidywania próbek jest popełniany tylko na krawędziach obrazu, oraz w tych jego fragmentach, które zawierają dużą ilość szczegółów (drobnych detali).



Rysunek 4-49. Prezentacja skuteczności **międzyobrazowej, jednokierunkowej** predykcji treści. Z uwagi na mechanizm działania predykcji próbek, niewielki błąd przewidywania treści można otrzymać także w przypadku tych fragmentów obrazu, które zawierają dużą ilość szczegółów (drobnych detali).

4.16. Międzyobrazowa predykcja treści – przewidywanie nie tylko z przeszłości, ale i przyszłości

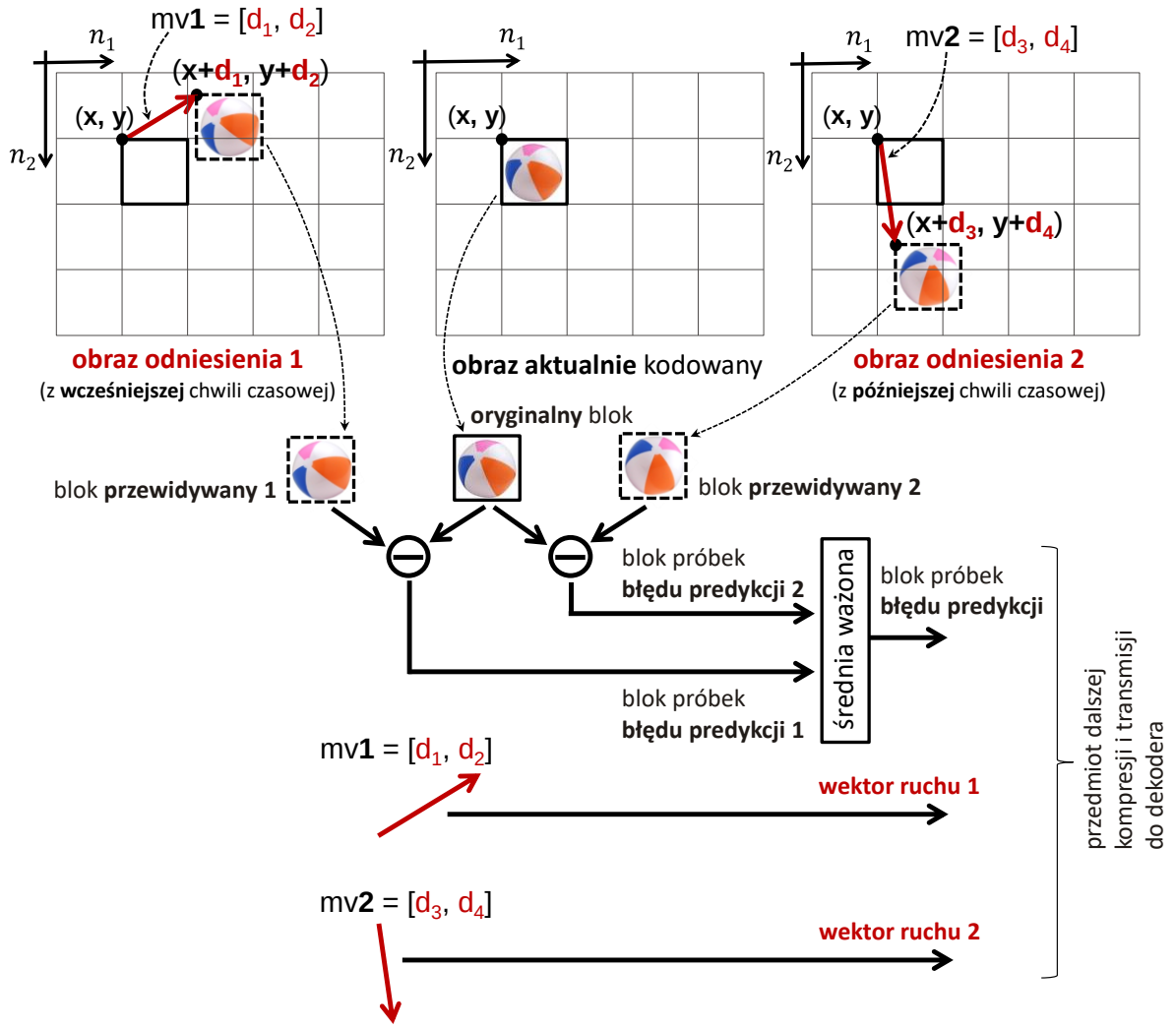
Omówiona w poprzednim punkcie jednokierunkowa predykcja wprzód treści kodowanego obrazu przypomina czynność prognozowania stanu pogody czy kursu walut na określony dzień, bazując na odpowiedniej informacji z dnia poprzedzającego. Również w tych przykładach, z dnia na dzień, nie zaobserwujemy raczej znaczących zmian, pomijając bardzo rzadkie ekstremalne sytuacje. Realizowane więc w ramach krótkich ram czasowych przewidywanie danych (np. pogoda, kursy walut czy treść kolejnych obrazów) będzie się zwykle odznaczać bardzo wysoką skutecznością.

Gdyby udało się jednak przewidywać na podstawie nie tylko tego, co już było, ale dodatkowo, w oparciu o znany stan z przyszłości… Czyli na przykład pogodę na dzisiaj, na podstawie jej stanu z wczoraj i z jutra. Taki sposób przewidywania danych, jak się można domyślać, byłby jeszcze dokładniejszy. Błąd predykcji danych byłby mniejszy. W przypadku kursu walut czy

pogody ten sposób predykcji jest oczywiście niemożliwy, bo nie znamy przyszłości. Ale można go zastosować w przypadku obrazów sekwencji, których treść, każdego z nich z osobna, jest koderowi z góry znana.

Idea takiego mechanizmu przewidywania treści została przedstawiona na poniższym rysunku (rysunek 4-50). Kodując dany blok koder próbuje znaleźć najbardziej podobną do niego treść niezależnie w dwóch różnych obrazach: w jednym z obrazów z wcześniejszej chwili czasowej (na rysunku 4-50 obraz ten został opisany jako „obraz odniesienia 1”) w stosunku do kodowanego obrazu oraz dodatkowo, w jednym z obrazów, który na osi czasu pojawia się później (jest to „obraz odniesienia 2”) niż obraz aktualnie kodowany. Ten sposób przewidywania treści wykorzystuje więc dwa **obrazy odniesienia**. Położenie w obrazach odniesienia tych najbardziej podobnych bloków (do bloku aktualnie kodowanego) jest więc wskazywane przez dwa niezależne wektory ruchu, oznaczone na rysunku jako $mv1$ i $mv2$, o wartościach składowych odpowiednio d_1 i d_2 oraz d_3 i d_4. Przywołane w poprzednim zdaniu najbardziej podobne bloki (dwa takie bloki) stanowią więc predykcję treści bloku, który aktualnie kodujemy. Te dwa niezależne sygnały predykcji podlegają ważonemu sumowaniu w celu wyznaczenia jednej predykcji kodowanej treści. Różnica pomiędzy kodowanym blokiem i tą predykcją daje już próbki błędu predykcji kodowanego bloku, który podlega w koderze dalszej kompresji.

Omawiana predykcja przewiduje więc treść bloków kodowanego obrazu z dwóch kierunków w czasie naraz, z chwili poprzedniej i chwili następnej. Dlatego jest ona nazywana **dwukierunkową predykcją międzyobrazową**. Przewidywanie, które odnosi się do treści obrazu odniesienia z wcześniejszej chwili czasowej realizuje **predykcję danych wprzód** (bo przewiduje treść obrazu, który jest późniejszy w czasie, czyli jest to przewidywanie do przodu). Z kolei to (przewidywanie), które korzysta z wiedzy o treści obrazu odniesienia, który jest w czasie później niż obraz kodowany dokonuje **wstecznej predykcji treści**, bo przewidywanie jest do tyłu, czyli z chwili późniejszej do chwili wcześniejszej. Jak zostanie wykazane w kolejnych punktach, dwukierunkowa predykcja treści istotnie zwiększa efektywność kodowania obrazów, jednak wiąże się z większym stopniem skomplikowania działania kodera i dekodera obrazu.



Rysunek 4-50. Dwukierunkowa międzyobrazowa predykcja treści obrazu. Aktualnie kodowany obraz jest przewidywany w oparciu o treść dwóch obrazów odniesienia.

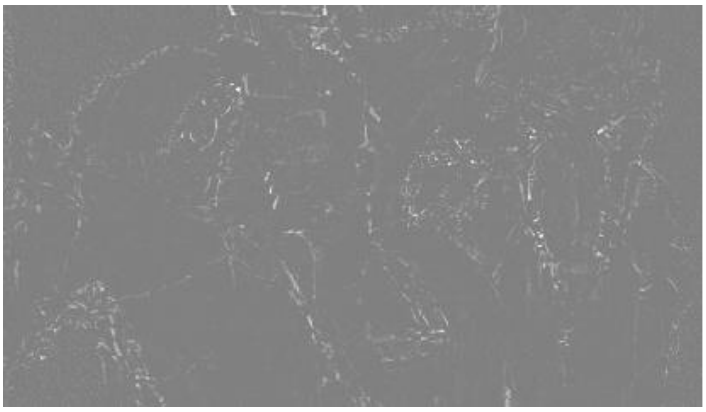
4.17. Predykcja międzyobrazowa jedno- i dwukierunkowa. Która jest lepsza?

Odpowiedź na to pytanie już padła przedstawiając ideę dwukierunkowej predykcji treści obrazu. Większą dokładność przewidywania zapewnia oczywiście predykcja, która bazuje na dwóch obrazach odniesienia. Jak duże są to różnice można się przekonać porównując ze sobą błędy przewidywania próbek obrazu, wraz z wartościami entropii H próbek błędu predykcji treści, jakie po zastosowaniu międzyobrazowej predykcji jednokierunkowej oraz dwukierunkowej zostały otrzymane dla wybranego obrazu sekwencji testowej (patrz rysunek 4-51).

obraz kodowany

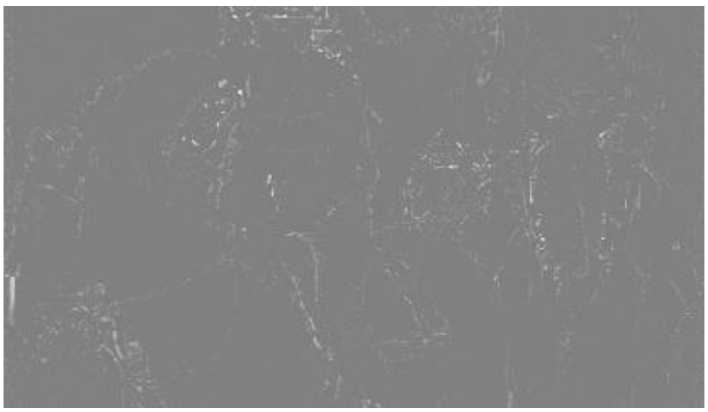
H = 7,2972 bita

obraz **błędu predykcji** próbek + 128
(predykcja międzyobrazowa **jednokierunkowa**)



H = 4,6160 bita

obraz **błędu predykcji** próbek + 128
(predykcja międzyobrazowa **dwukierunkowa**)



H = 4,2784 bita

Rysunek 4-51. Porównanie skuteczności predykcji **międzyobrazowej, jedno-** i **dwukierunkowej**.

W tym punkcie postaramy się jednak dokładniej wyjaśnić, czemu zawdzięczamy tę wyższość **predykcji dwukierunkowej**. A powodów jest co najmniej kilka.

Po pierwsze, obecny w obrazie szum. Jego źródłem są występujące w układach elektronicznych kamery sygnały szumu, które przenikają niejako (dodają się) do rejestrowanego przez kamerę obrazu. Szum w obrazie można zaobserwować szczególnie łatwo, jeśli ogląda się obraz przy odpowiednim powiększeniu. Można to zobaczyć na zamieszczonym na rysunku 4-52 przykładzie.

Rysunek 4-52. Ilustracja obecności szumu w obrazie. Szum ten jest wynikiem niedoskonałości działania kamery, która rejestruje obraz.

Szum ma to do siebie, że przyjmuje zupełnie losowe wartości, inne w poszczególnych fragmentach obrazu. Dlatego nie sposób skutecznie przewidzieć jego próbek (gdyby udało się przewidzieć wartości takiego sygnału, to oznaczałoby, że nie jest on szumem). Szumy występują więc w kodowanym bloku obrazu, jak również w bloku predykcji, który pochodzi z obrazu odniesienia. Jednak nie są one takie same, identyczne. Są od siebie różne. Zwiększa to różnicę między tymi

dwoma blokami powodując, że wartości próbek błędu predykcji przyjmują większe wartości. Tak jest w przypadku predykcji treści bloku, realizowanej z jednego kierunku w czasie.

Remedium na ten problem jest usunięcie lub choćby częściowe ograniczenie poziomu szumu w bloku, który stanowi predykcję dla aktualnie kodowanych danych. I właśnie takie ograniczenie ma miejsce w przypadku predykcji dwukierunkowej. Tutaj, niezależnie od siebie wyznacza się dwa bloki predykcji, bo z wykorzystaniem dwóch różnych obrazów odniesienia, po czym dokonuje się ważonej sumy tych dwóch bloków. Wynik tej ważonej sumy stanowi już ostateczną predykcję treści kodowanego bloku. Ponieważ operacja uśrednienia wartości próbek działa jak prosty filtr dolnoprzepustowy, to w drodze tej operacji próbki szumu ulegają pewnemu rozmyciu, przez co sygnał szumu jest redukowany. Dzięki temu rozmyciu, uzyskiwany w drodze dwukierunkowej predykcji błąd przewidywania treści jest znacznie mniejszy.

Drugi z powodów ma związek z tym, że jedne obiekty zasłaniają inne, częściowo lub całkowicie. Wynika to z ruchu obiektów rejestrowanej sceny. Fragment, który kodujemy może być więc niewidoczny w obrazie odniesienia, na podstawie którego przewidujemy bieżącą treść. Jednak w przypadku dwukierunkowej predykcji mamy jeszcze drugi obraz odniesienia, który pochodzi z innej chwili czasowej niż poprzedni obraz (odniesienia). Ta zasłonięta wcześniej treść może być w tym drugim obrazie (odniesienia) już dobrze widoczna. Efektem będzie oczywiście dokładniejsza predykcja kodowanej treści.

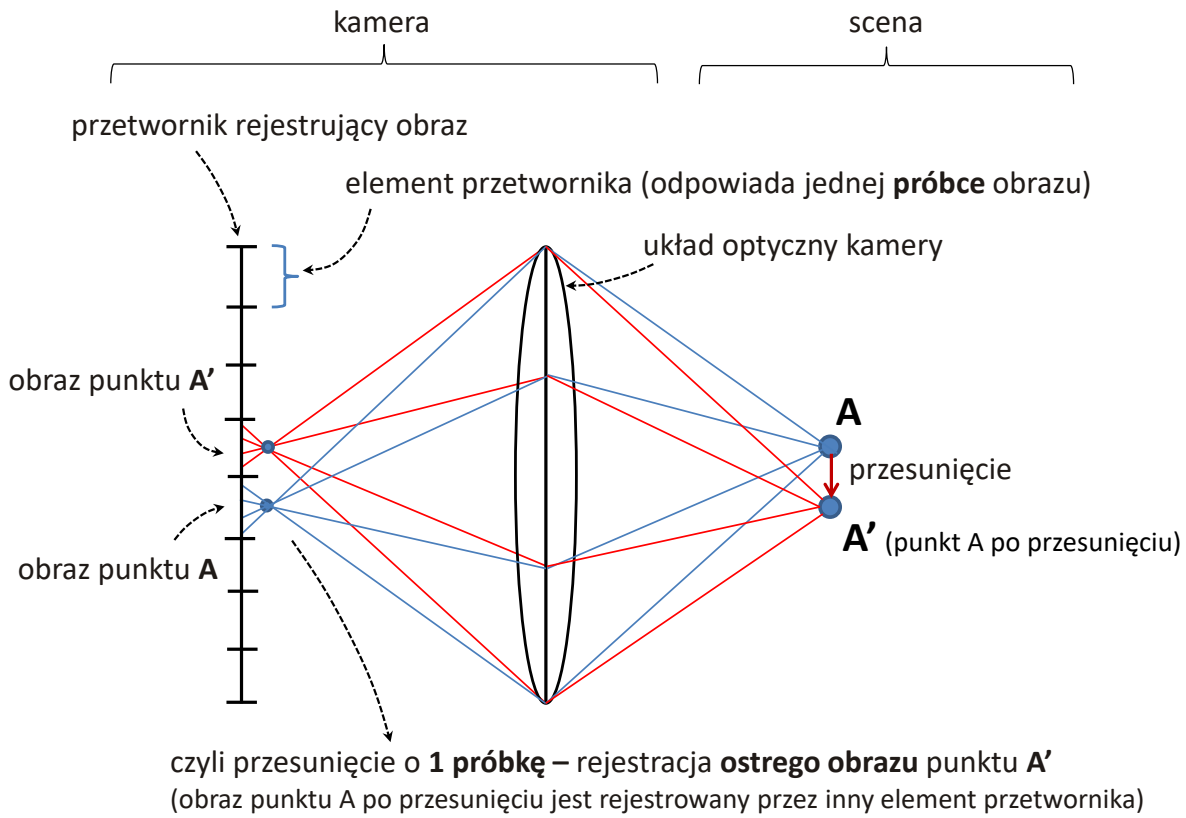
Oprócz wymienionych sytuacji może wystąpić jeszcze szereg innych przypadków. Z obrazu na obraz zmienić się może sposób oświetlenia sceny. Poruszające się obiekty sceny mogą się zmieniać w czasie w inny sposób niż wynika to z prostego modelu ruchu translacyjnego na płaszczyźnie obrazu (np. mogą doznawać innych przekształceń geometrycznych), itd.. Takie przypadki lepiej w ogólności „obsłuży” predykcja dwukierunkowa niż jednokierunkowa.

4.18. Sposób na dodatkowe zwiększenie dokładności metody – estymacja ruchu z ułamkową dokładnością

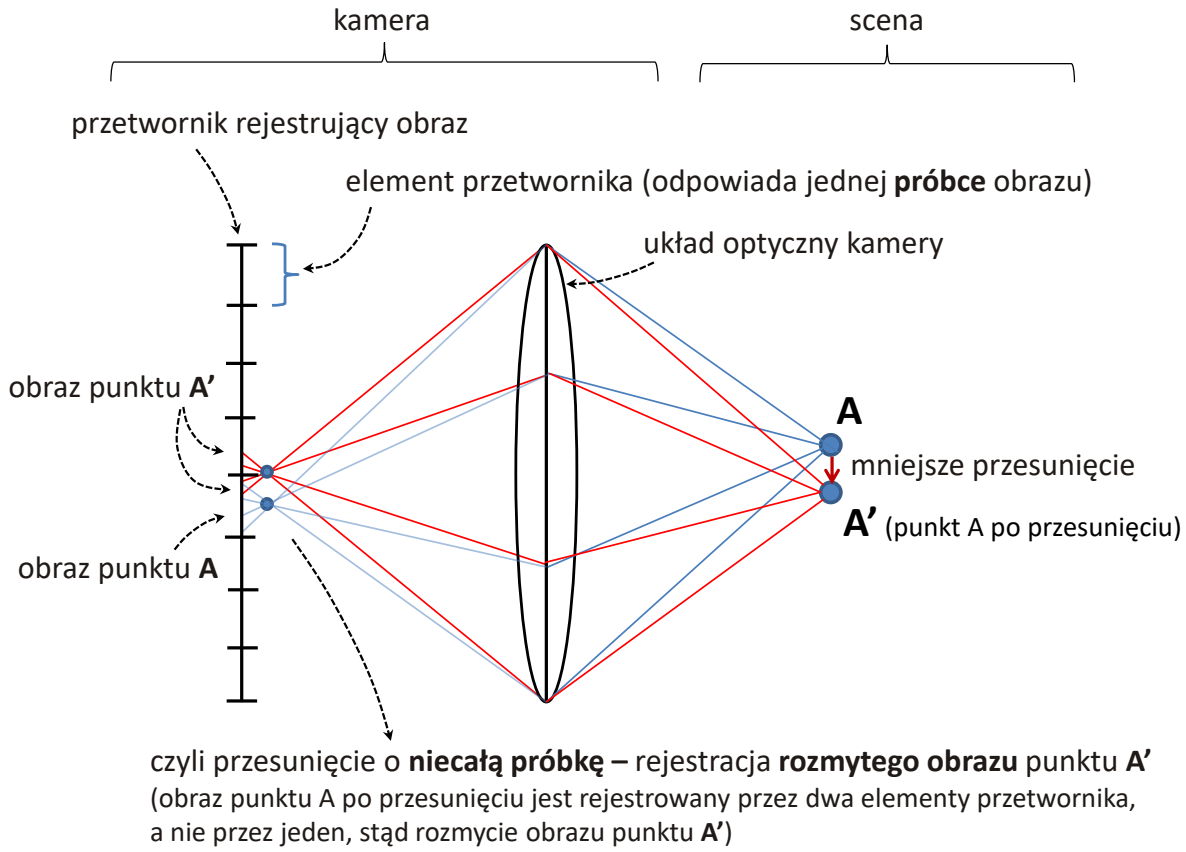
4.18.1. Wprowadzenie teoretyczne

W dotychczasowych rozważaniach na temat estymacji ruchu milcząco przyjmowano założenie, że obiekty sceny mogą się przemieszczać tylko o dystans, który w zarejestrowanym obrazie odpowiada przesunięciu o całkowitą liczbę próbek. Czyli np. o jedną próbkę, pięć próbek, itd., w kierunku pionowym czy poziomym, ale nigdy o ułamkową liczbę próbek.

Ruch obiektów jest jednak całkowicie niezależny od punktów, w których kamera realizuje próbkowanie obrazu sceny. Nie ma więc pewności, że za każdym razem obiekty sceny będą się przesuwać dokładnie do tych punktów, które rejestruje kamera. W rzeczywistości przesunięcia te mogą stanowić ułamkową cześć okresu próbkowania obrazu (odstępu między próbkami obrazu). Zamieszczone poniżej rysunki pozwolą lepiej zrozumieć omawiane zjawisko.



Rysunek 4-53. Ilustracja mechanizmu rejestracji obrazu w kamerze. Przesunięcie punktu sceny powoduje rejestrację jego obrazu przez inny element (elementy) światłoczułego przetwornika kamery. W tym przypadku wielkość przesunięcia punktu **A** w scenie jest taka, że odpowiada ona przesunięciu obrazu tego punktu **o 1 próbkę**. Idea wykonania rysunku została zaczerpnięta z prezentacji Krzysztofa Wegnera pt. „Wprowadzenie teoretyczne do przestrzeni pola światła”.



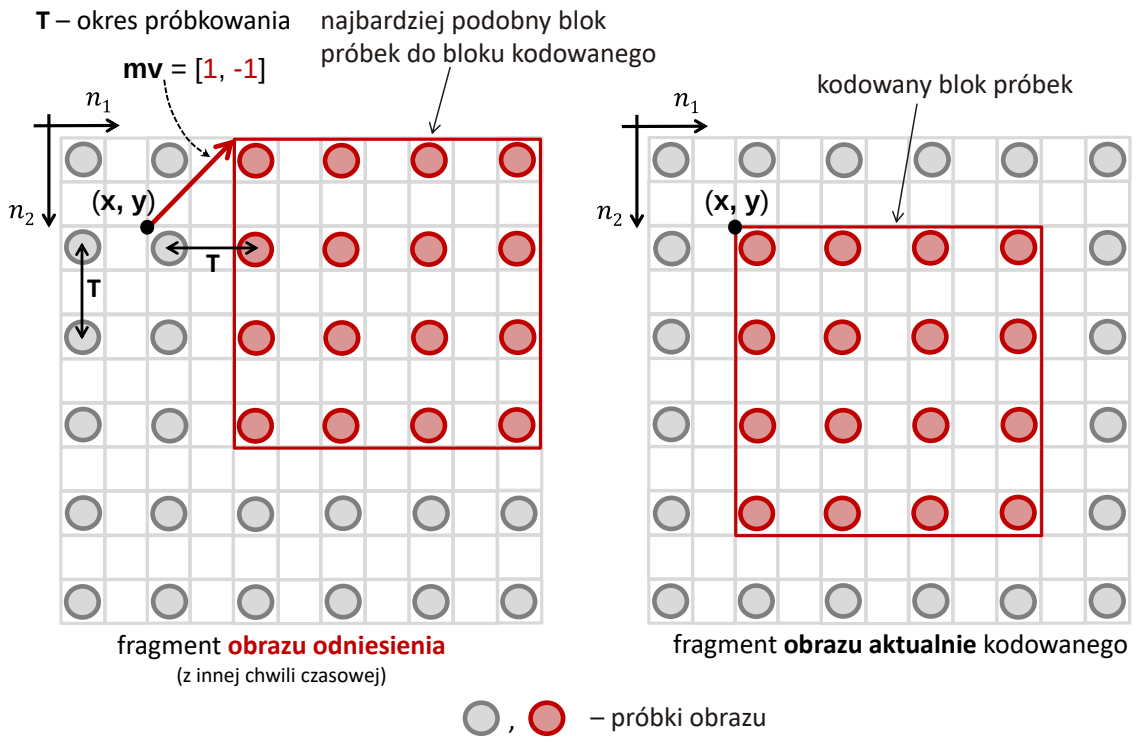
Rysunek 4-54. Ilustracja mechanizmu rejestracji obrazu w kamerze. Przesunięcie punktu sceny powoduje rejestrację jego obrazu przez inny element (elementy) światłoczułego przetwornika kamery. W tym przypadku wielkość przesunięcia punktu **A** w scenie jest taka, że odpowiada ona przesunięciu obrazu tego punktu **o mniej niż 1 próbkę**. Idea wykonania rysunku została zaczerpnięta z prezentacji Krzysztofa Wegnera pt. „Wprowadzenie teoretyczne do przestrzeni pola światła”.

W bardzo ogólnym ujęciu można powiedzieć, że poszczególne punkty sceny wysyłają do kamery promienie światła. Światło to jest faktycznie emitowane przez dany punkt sceny, bądź jest przez niego tylko odbijane. Zadaniem tradycyjnej kamery jest zarejestrowanie informacji o tym, jaka jest sumaryczna intensywność wszystkich promieni światła, które wychodzą z danego punktu sceny, w danej chwili czasowej. Celem pobrania tej informacji promienie światła zostają odpowiednio skupione przez układ optyczny kamery, a wynik tego skupienia, czyli całościowe światło danego punktu jest rejestrowane przez światłoczuły przetwornik zlokalizowany w kamerze (za jej układem optycznym). Przy odpowiednich ustawieniach układu optycznego kamery wszystkie promienie światła, które wychodzą z danego punktu sceny daje się skupić na jednym elemencie światłoczułego przetwornika, dzięki czemu obraz punktu sceny zostaje zarejestrowany jako ostry. Ten uproszczony mechanizm działania kamery dla punktu **A** sceny ilustrują powyższe rysunki.

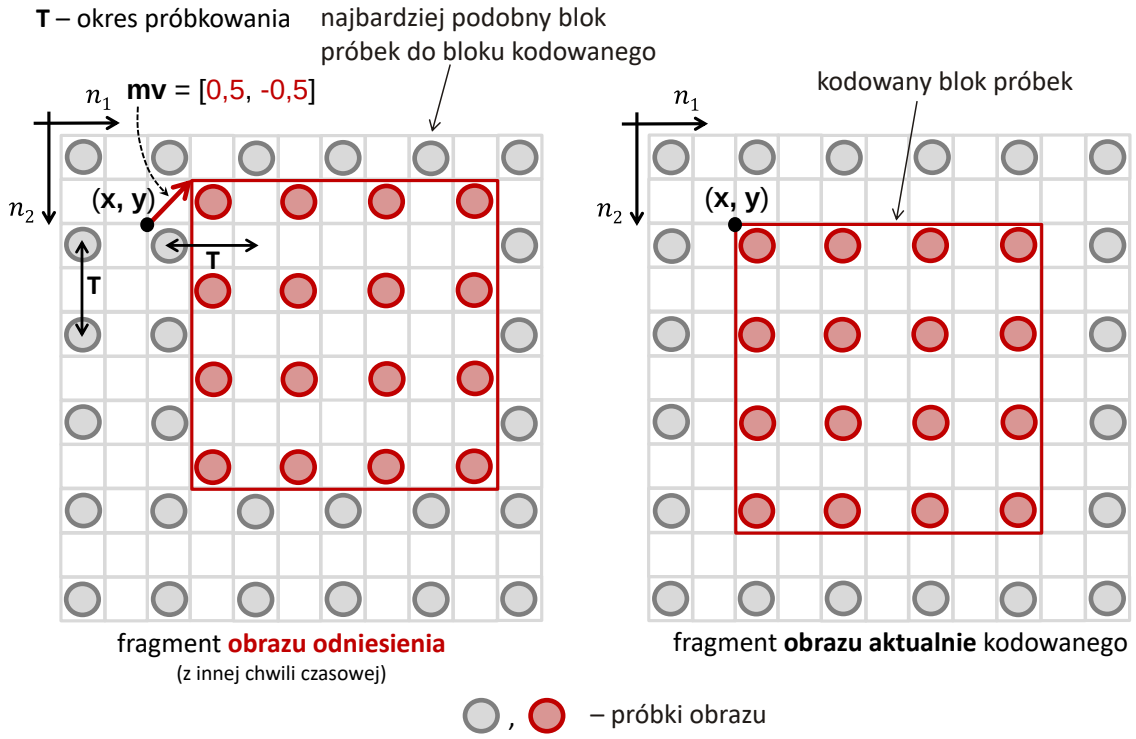
W przypadku przesunięcia się punktu **A** sceny do nowego położenia, które zostało oznaczone jako **A'** (patrz rysunek 4-53), promienie jego światła zostaną po przejściu przez układ optyczny skupione na innym, ale wciąż jednym elemencie przetwornika kamery. W tym przypadku jest to element, który bezpośrednio sąsiaduje z tym poprzednim, co w obrazie odpowiada przesunięciu się punktu **A** dokładnie o jedną próbkę obrazu. Obraz punktu **A'** będzie więc po rejestracji również ostry.

Ale proszę zobaczyć, co się stanie, jeśli przesunięcie punktu sceny będzie na przykład trochę mniejsze, np. takie jak rysunku 4-54. Wynikiem takiej zmiany położenia punktu będzie „rozlanie” się związanego z nim światła na dwóch elementach przetwornika, a nie tak jak wcześniej rejestracja na jednym elemencie, przez co obraz punktu sceny nie będzie już ostry, tylko w odpowiedni sposób rozmyty. Owo rozmycie jest właśnie wynikiem przesunięcia się punktu sceny o wielkość, która w zarejestrowanym obrazie odpowiada przesunięciu o ułamkową część odstępu pomiędzy próbkami obrazu. W przedstawionym na rysunku 4-54 przypadku jest to przesunięcie o mniej niż jedną próbkę obrazu. Gdyby to samo rozpatrywać na poziomie obiektów sceny, a nie pojedynczych punktów, to skutkiem „ułamkowego” przesunięcia będzie właściwe rozmycie obiektu sceny. Sposób i wielkość rozmycia bezpośrednio zależą od wielkości przemieszczenia się obiektu w scenie.

Omówiona do tej pory estymacja ruchu nie uwzględnia niestety tego fenomenu. Oczywistym tego skutkiem jest mniejsza efektywność metody estymacji ruchu dla wszystkich tych obiektów sceny, które doznają w obrazie przesunięć o ułamkową liczbę próbek. Jednak można temu dość łatwo zaradzić. Skoro niektóre przesunięcia prowadzą do „rozmycia” obiektu w obrazie zarejestrowanym w bieżącej chwili czasowej, to efekt tego rozmycia można dość łatwo zasymulować, realizując dolnoprzepustową filtrację próbek obrazu odniesienia (czyli obrazu z innej chwili), które opisują ten obiekt. W drodze ważonej sumy pewnej liczby próbek, które są obecne w obrazie odniesienia (co odpowiada właśnie jakiejś filtracji dolnoprzepustowej próbek) wylicza się zupełnie nowe próbki, które dużo lepiej reprezentują rozmyty obiekt, obecny w bieżącym obrazie. Z ideowego punktu widzenia filtracja ta odpowiada wyliczeniu nowych próbek obrazu, które w obrazie znajdują się gdzieś pomiędzy tymi próbkami, które wcześniej zarejestrowała kamera (patrz rysunek 4-57). Biorąc ten fakt pod uwagę, jak również sam sposób wyliczania nowych wartości operacja ta jest niczym innym jak **interpolacją próbek** obrazu, na podstawie znanych wartości próbek sąsiednich. Sposób przeprowadzenia interpolacji próbek determinuje wielkość i rodzaj rozmycia obrazu obiektu, co odpowiada ściśle określonemu przemieszczeniu się obiektu w rejestrowanej przez kamerę scenie. Estymacja ruchu z interpolacją próbek pozwala więc dokładniej przewidywać kodowane fragmenty obrazu, ponieważ oprócz standardowych przesunięć o całkowitą liczbę próbek (jak na rysunku 4-55) uwzględnia przypadki przesunięć obiektów o liczbę próbek niecałkowitą (patrz rysunek 4-56). Nie trzeba jednak przekonywać, że zwiększa to złożoność operacji przewidywania treści.



Rysunek 4-55. Estymacja ruchu z całopunktową dokładnością. W tym przypadku nie zachodzi potrzeba interpolacji próbek obrazu.

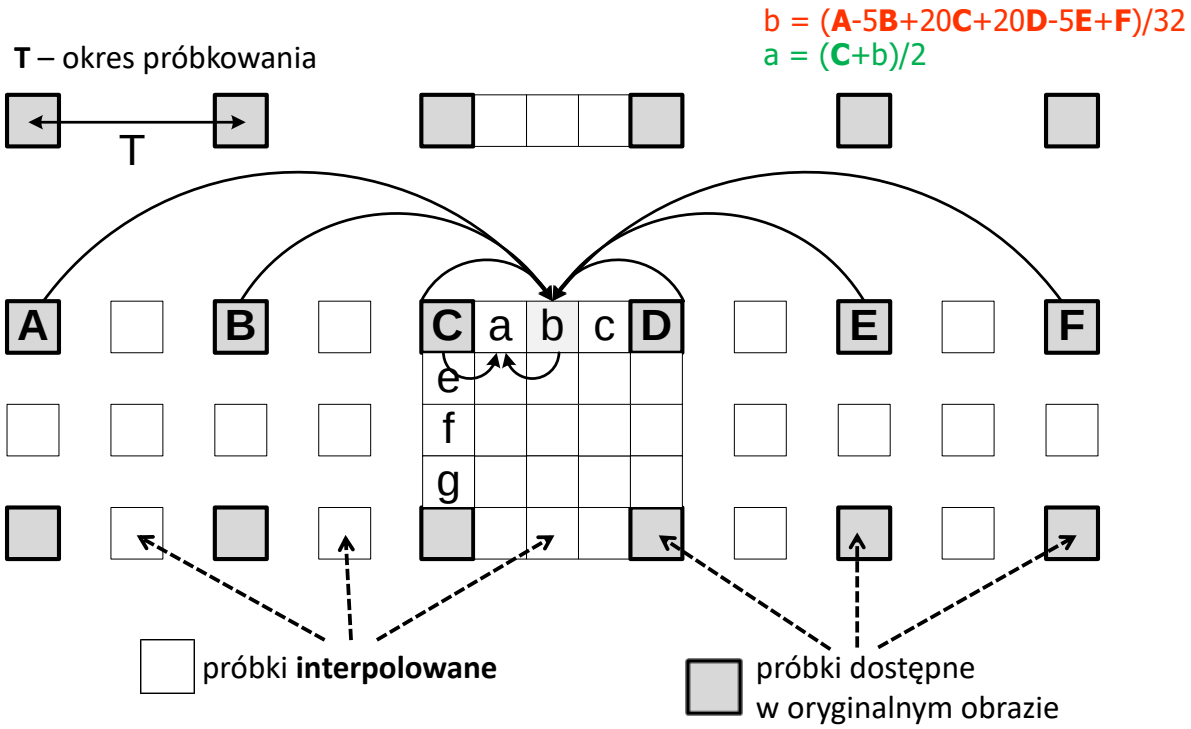


Rysunek 4-56. Estymacja ruchu z **półpunktową** dokładnością. W tym przypadku konieczna jest interpolacja próbek obrazu (czerwone próbki w obrazie odniesienia) na podstawie próbek, które zarejestrowała kamera (próbki szare).

4.18.2. Praktyczna realizacja estymacji ruchu z ułamkową dokładnością

W praktyce, **interpolacja próbek** jest realizowana z użyciem wcześniej zdefiniowanych dolnoprzepustowych filtrów. Ich działanie sprowadza się do policzenia ważonej sumy określonego zestawu próbek, które są dostępne w obrazie odniesienia (czyli faktycznie zostały zarejestrowane przez kamerę). Dokładny sposób ważenia próbek, oraz liczba próbek, które w danym etapie obliczeniowym podlegają temu ważeniu jednoznacznie definiują stosowany **filtr interpolacyjny**.

Zwykle stosuje się więcej niż jeden filtr. Rodzaj stosowanych filtrów jest zawsze kompromisem pomiędzy uzyskiwanymi wynikami interpolacji próbek (co bezpośrednio przekłada się na efektywność metody estymacji ruchu) a złożonością obliczeniową algorytmu przewidywania bloków obrazu. Współcześnie stosowane kodery obrazu (np. koder AVC) dokonują interpolacji wartości próbek, które znajdują się dokładnie w połowie odległości dwóch dostępnych próbek (patrz próbki **b** i **f** na rysunku 4-57), oraz w 1/4 i 3/4 tej odległości (patrz próbki **a** i **c** na rysunku 4-57). Mówi się więc w tym miejscu o interpolacji próbek z dokładnością odpowiednio **pół-punktową** i **ćwierć-punktową**. W przywołanym wcześniej kodowaniu AVC próbki położone tak jak próbka **b** (patrz rysunek) są wyznaczane jako ważona suma 6 dostępnych próbek: 3 zlokalizowanych na lewo od próbki interpolowanej i trzech znajdujących się po jej prawej stronie. Sposób ważenia jest ściśle określony – taki jak na przedstawionym rysunku. Próbka **f** będzie wyznaczana według tej samej ogólnej reguły, jednak ważeniu będą podlegać sąsiednie próbki, które znajdują się w tej samej kolumnie (a nie tak jak wcześniej w tym samym wierszu), co interpolowana próbka. Próbki **a**, **c**, **e**, **g** są wynikiem prostego uśrednienia sąsiadujących próbek z pozycji cało-punktowej oraz pół-punktowej.



Rysunek 4-57. Ilustracja mechanizmu interpolacji próbek obrazu. Idea wykonania rysunku zaczerpnięta z pracy [Rao06].

4.18.3. Jak dokładniejsza estymacja ruchu wpływa na efektywność kodera obrazu?

Zwiększenie dokładności estymowania ruchu poprawia zdolności kodeka do przewidywania treści, co skutkuje oczywiście większą efektywnością kompresji obrazów. Ale zwiększa się również złożoność obliczeniowa kodera i dekodera. Dlatego w tym miejscu można się zastanawiać jak duży jest to zysk? I czy faktycznie uzasadnia on dodatkowe nakłady obliczeniowe w koderze i dekoderze obrazu?

Dokładne odpowiedzi na powyższe pytania zależą od szeregu czynników. Z jaką technologią kompresji obrazów mamy do czynienia (technologia starsza, np. MPEG-2, czy nowsza, np. AVC), jaki mechanizm wyboru trybów stosuje koder, czy wreszcie, jaki jest stopień optymalizacji kodu programu kodera i dekodera? Żeby jednak przybliżyć czytelnikowi możliwe odpowiedzi autor przeprowadził dedykowany eksperyment z użyciem modelowego oprogramowania **JM 18.5** kodeka **AVC** [AVCSoft]. Celem tego eksperymentu była ocena efektywności kompresji danych, jak również obliczeniowej złożoności kodera i dekodera obrazów w dwóch przypadkach:

1) Kiedy kodowanie obrazów korzysta z wyników estymacji ruchu, która jest realizowana z zaledwie całkowitoliczbową dokładnością;

2) Kiedy koder estymuje ruch w sekwencji z dokładnością większą niż całkowitoliczbowa, czyli z dokładnością do 1/2 oraz 1/4 okresu próbkowania obrazu.

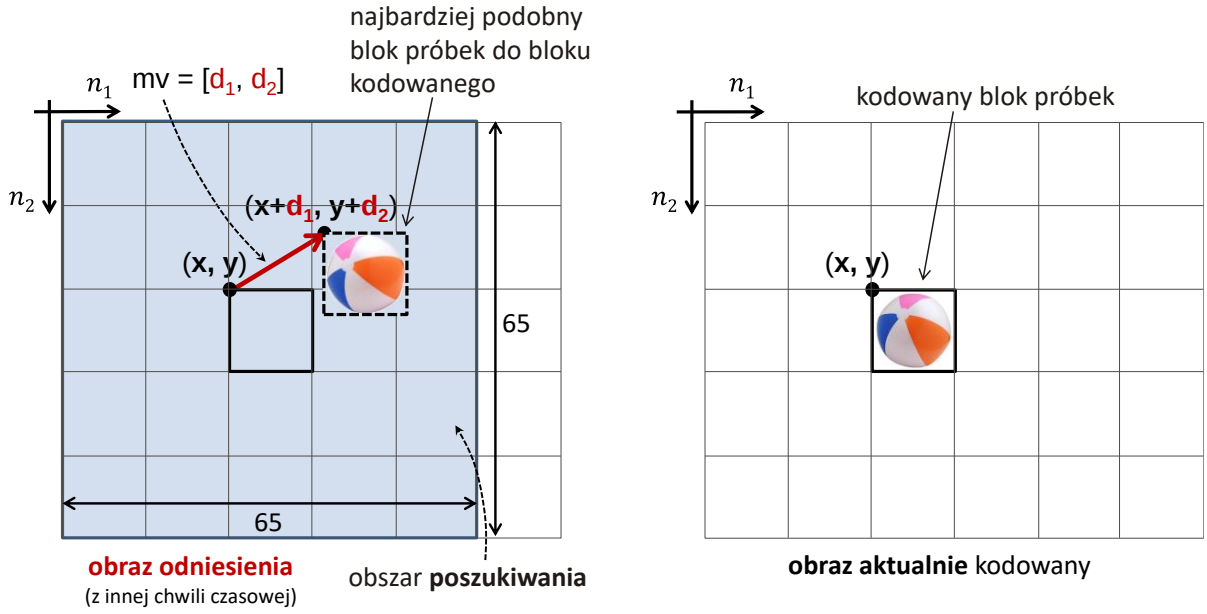
Kodując obrazy przy pomocy puli narzędzi kompresji, która jest przewidziana dla telewizji cyfrowej otrzymane zostały następujące rezultaty. Zwiększenie dokładności estymacji ruchu stało się przyczyną aż **3-krotnego** wydłużenia czasu wyznaczania wektorów ruchu w koderze. Estymacja ruchu stanowi bardzo dużą część obliczeń kodera, dlatego przełożyło się to finalnie na **1,6-krotny** wzrost złożoności kodera obrazu. Jest to oczywiście bardzo znaczący wzrost. Dużo optymistyczniej wyglądają dane, jakie otrzymano dla dekodera. Tutaj nie ma estymacji ruchu, natomiast pojawia się etap interpolacji obrazu w przypadku stosowania wektorów ruchu, które wskazują na przesunięcia o ułamkową liczbę próbek. Wyniki pokazują, że ta dodatkowa interpolacja próbek obrazu w dekoderze wydłuża czas jego działania o około **10%**. Dzięki tym dodatkowym nakładom obliczeniowym otrzymujemy jednak bardzo znaczący (jak na możliwości kompresji danych) zysk w postaci redukcji zakodowanego strumienia bitowego o **10% – 15%**, utrzymując taką samą, jak wcześniej, jakość zakodowanych obrazów.

4.19. Standardowy sposób estymacji ruchu – złożoność obliczeniowa

Estymacja ruchu obiektów sceny, która jest realizowana z użyciem metody pasowania bloków intuicyjnie wydaje się bardzo prosta. Cała rzecz sprowadza się do „prostego” porównywania wartości próbek dwóch bloków: aktualnie kodowanego i bloku odniesienia. Jednak w praktyce jest dokładnie odwrotnie. Ilość obliczeń, jaką pochłania operacja porównywania bloków jest wręcz ogromna! Można się o tym łatwo przekonać, analizując poniższy przykład.

Wyobraźmy sobie, że ruch jest estymowany tylko i wyłącznie w ramach bloków o stałej wielkości 16x16 próbek składowej luminancji, w sekwencji obrazów o rozdzielczości przestrzennej 1920 x 1080 (czyli full HD), których w czasie jednej sekundy jest 50. Dodatkowo, przyjmijmy założenie, że estymujemy ruch korzystając z jednego tylko obrazu odniesienia. Poszukiwanie najbardziej podobnego (do aktualnie przetwarzanego bloku) bloku próbek w całym obrazie

odniesienia byłoby obliczeniowo ekstremalnie trudne. Dlatego w tym miejscu dokonamy bardzo istotnego uproszczenia, zgodnie z którym, tego najbardziej podobnego bloku będziemy szukać tylko w niewielkim fragmencie obrazu odniesienia o wielkości 65x65 próbek obrazu, tak jak przedstawiono to na poniższym rysunku.



Rysunek 4-58. Estymacja ruchu realizowana w ramach z góry ustalonego **obszaru poszukiwania**. Na rysunku przyjęto, że obszar poszukiwania ma wielkość 65x65 próbek obrazu.

Fragment ten tworzy tzw. **obszar poszukiwania** dla algorytmu estymacji ruchu, którego środek wypada w obrazie odniesienia w punkcie o współrzędnych (x, y). Dla przypomnienia, w tych samych współrzędnych jest „zakotwiczony” lewy górny narożnik aktualnie kodowanego bloku próbek w kodowanym obrazie. Obszar poszukiwania jest więc tym rejonem (fragmentem) obrazu odniesienia, który znajduje się w tej samej części obrazu co aktualnie kodowany blok w obrazie kodowanym. Podstawą przyjętego uproszczenia jest obserwacja, iż w bardzo małych odstępach czasu, np. takich jak odległość dwóch kolejnych obrazów sekwencji, obiekty sceny nie doznają zwykle znacznych przesunięć. Dlatego zazwyczaj i tak nie ma sensu realizować poszukiwania poza tym obszarem. Jednak nawet w przypadku tak małego obszaru poszukiwania różnych bloków 16x16, których lewy górny narożnik zawiera się w tym obszarze jest całkiem dużo, bo **65 x 65**. Każdy z tych bloków zawiera **16 x 16** próbek, co daje w sumie liczbę **16 x 16 x 65 x 65 = 1 081 600** porównań odpowiadających sobie wartości próbek (w bloku odniesienia i bloku kodowanym) dwóch bloków. Ta niemała przecież liczba porównań dwóch wartości daje w rezultacie dopiero jeden wektor ruchu, który umożliwia wydajną predykcję treści zaledwie jednego bloku 16x16. Na tym oczywiście jeszcze nie koniec. Powyższe czynności należy przecież powtórzyć dla pozostałych bloków 16x16 obrazu. W sumie tych bloków w obrazie full HD jest aż **8100**, co po uwzględnieniu ostatnio otrzymanej liczby porównań daje pokaźną już liczbę **8 760 960 000** porównań próbek (jest to wynik iloczynu poprzedniej liczby porównań z wartością 8100). Tyle porównań wartości dwóch próbek musi wykonać koder w ramach jednego tylko obrazu sekwencji. Przyjęliśmy, że w czasie jednej sekundy obrazów jest **50**, więc iloczyn **50 x 8 760 960 000 ≈ 4,38 x 10^{11}** stanowi liczbę porównań dwóch próbek obrazu, jaką wykonuje koder w czasie jednej sekundy. Otrzymana liczba jest więc ogromna. Wykonanie przez koder obrazu operacji porównania wartości próbek wymaga uprzedniego załadowania ich z określonego obszaru pamięci (co też się wiąże z jakimś czasem), tworząc algorytm estymacji ruchu bardzo, ale to bardzo obliczeniowo złożonym.

Proszę zauważyć, że tę gigantyczną liczbę porównań próbek otrzymano mimo przyjęcia już na starcie szeregu ograniczeń, uproszczeń, które mocno zmniejszają tę liczbę porównań. Jednocześnie, obniżają efektywność przewidywania treści. Przypomnijmy raz jeszcze. Estymacja ruchu ma miejsce w ramach mocno zawężonego obszaru poszukiwania (obszar o wielkości 65x65 próbek), jest realizowana w blokach o jednym, z góry ustalonym rozmiarze (bloki 16x16) i zakłada dodatkowo, że obiekty mogą się przesuwać jedynie o całkowitą liczbę próbek obrazu, czyli z pominięciem przesunięć o ułamkową część odstępu między próbkami w obrazie. Istotnym uproszczeniem jest ponadto predykcja treści z jednego tylko kierunku w czasie, czyli z użyciem jednego obrazu odniesienia. Jest więc oczywiste, że rezygnacja ze wspomnianych ograniczeń pozwoliłaby uzyskać jeszcze wyższą efektywność przewidywania treści, jednak ceną byłaby znacznie wyższa złożoność algorytmu. Z niektórych z tych ograniczeń warto jest ostatecznie zrezygnować, co i tak czyni operację estymacji ruchu wielkim wyzwaniem, również dla współczesnych, wysokowydajnych procesorów i układów scalonych.

Z punktu więc widzenia zastosowania praktycznego metody w koderach obrazu kluczowe stały się rozwiązania, które istotnie zredukują tę ogromną złożoność obliczeniową algorytmu.

4.20. Szybkie metody estymacji ruchu

Olbrzymia złożoność obliczeniowa oryginalnej metody estymacji ruchu zrodziła konieczność znaczącego uproszczenia metody przed jej właściwym użyciem w koderze obrazu. Potrzeba redukcji złożoności metody stała się motywacją dla badań w zakresie suboptymalnych metod wyznaczania wektorów ruchu. Suboptymalnych, to znaczy zapewniających dobre bądź bardzo dobre (chociaż nie zawsze najlepsze) rezultaty estymacji ruchu przy jednoczesnej silnej redukcji złożoności algorytmu wyznaczania wektorów ruchu.

W toku badań opracowanych zostało wiele takich uproszczonych metod [Salom06, Salom10]. Dobrym przykładem są tutaj trzy metody: 1) **dwuwymiarowe poszukiwanie logarytmiczne** (ang. two-dimensional logarithmic search), 2) **algorytm trzech kroków** (ang. three-step search) oraz 3) **poszukiwanie hierarchiczne** (ang. hierarchical search). Literatura zna oczywiście również inne metody, stanowiące chociażby pewną modyfikację czy rozwinięcie wskazanych metod, np. [Zhu00], czy [Purn12]. Jednak zdaniem autora przyswojenie sobie idei trzech wymienionych metod pozwoli już dobrze zrozumieć klucz, który otwiera drogę do szybkich algorytmów estymacji ruchu.

Tym kluczem jest silne zawężenie zbioru bloków obrazu odniesienia, z którymi to blokami porównuje się blok aktualnie kodowany. Wspominane bloki obrazu odniesienia wybiera się w ściśle określony sposób, który jest charakterystyczny dla danej metody estymacji ruchu. Działanie metody rozpoczynamy od zdefiniowania obszaru poszukiwania. Ma on rozmiar $(2 \cdot d + 1) \times (2 \cdot d + 1)$ próbek obrazu (gdzie d jest parametrem metody), przy czym jego środek ma takie same współrzędne w obrazie odniesienia, co koordynaty lewego górnego narożnika aktualnie kodowanego bloku próbek w aktualnie kodowanym obrazie. Wielkość d jest więc parametrem metody – dla potrzeb naszego przykładu załóżmy że ma on wartość $d = 8$. Znając już rozmiar obszaru poszukiwania bloków działanie pierwszej z metod sprowadza się do wykonania następujących etapów obliczeniowych:

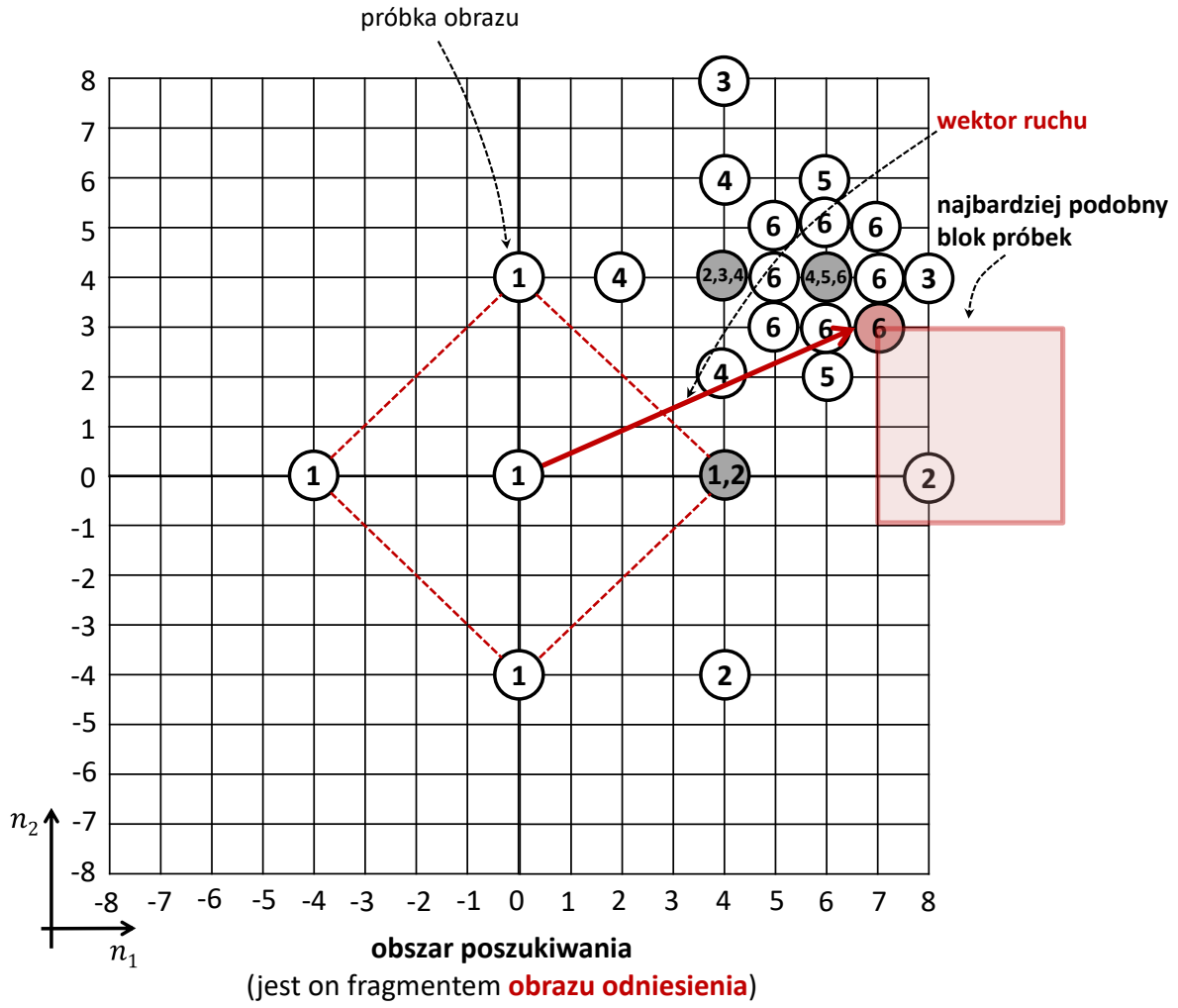
Etap 1: Począwszy od środka obszaru poszukiwania wybiera się pięć punktów w taki sposób, że tworzą one wzorzec/znak +. Żeby zgodnie z przyjętym wzorcem rozlokować te pięć punktów równomiernie wewnątrz obszaru poszukiwania to przesuwamy się względem jego środka (gdzie znajduje się już punkt **centralny**) o $s = 2^{\lfloor log_2(d) \rfloor - 1}$ w lewo, w prawo, w górę oraz w dół. Wartość

s jest więc rozmiarem kroku przesunięcia, który zależy od parametru d. Z uwagi na założoną wcześniej wartość $d = 8$ rozmiar kroku będzie w naszym przypadku równy $s = 4$. Otrzymane punkty (oznaczone na rysunku 4-59 numerem 1) określają położenie lewego-górnego narożnika bloków próbek, które należy porównać z aktualnie kodowanym blokiem. Na tym etapie mamy więc 5 porównań par bloków.

Etap 2: Z wcześniejszego porównania bloków wybieramy ten blok obrazu odniesienia, który w największym stopniu przypomina blok, który aktualnie kodujemy. Wspomniany blok obrazu odniesienia nazwiemy w tym miejscu „najlepszym” blokiem. Lewy-górny narożnik tego bloku stanowi **centrum** dla wyboru nowych pięciu punktów. W tym wyborze stosuje się ten sam, co wcześniej, rozmiar kroku s, jeśli „najlepszy” blok był związany z punktem znaku +, który nie był punktem **centralnym** tego wzorca lub s jest dwukrotnie zmniejszane, jeśli było inaczej.

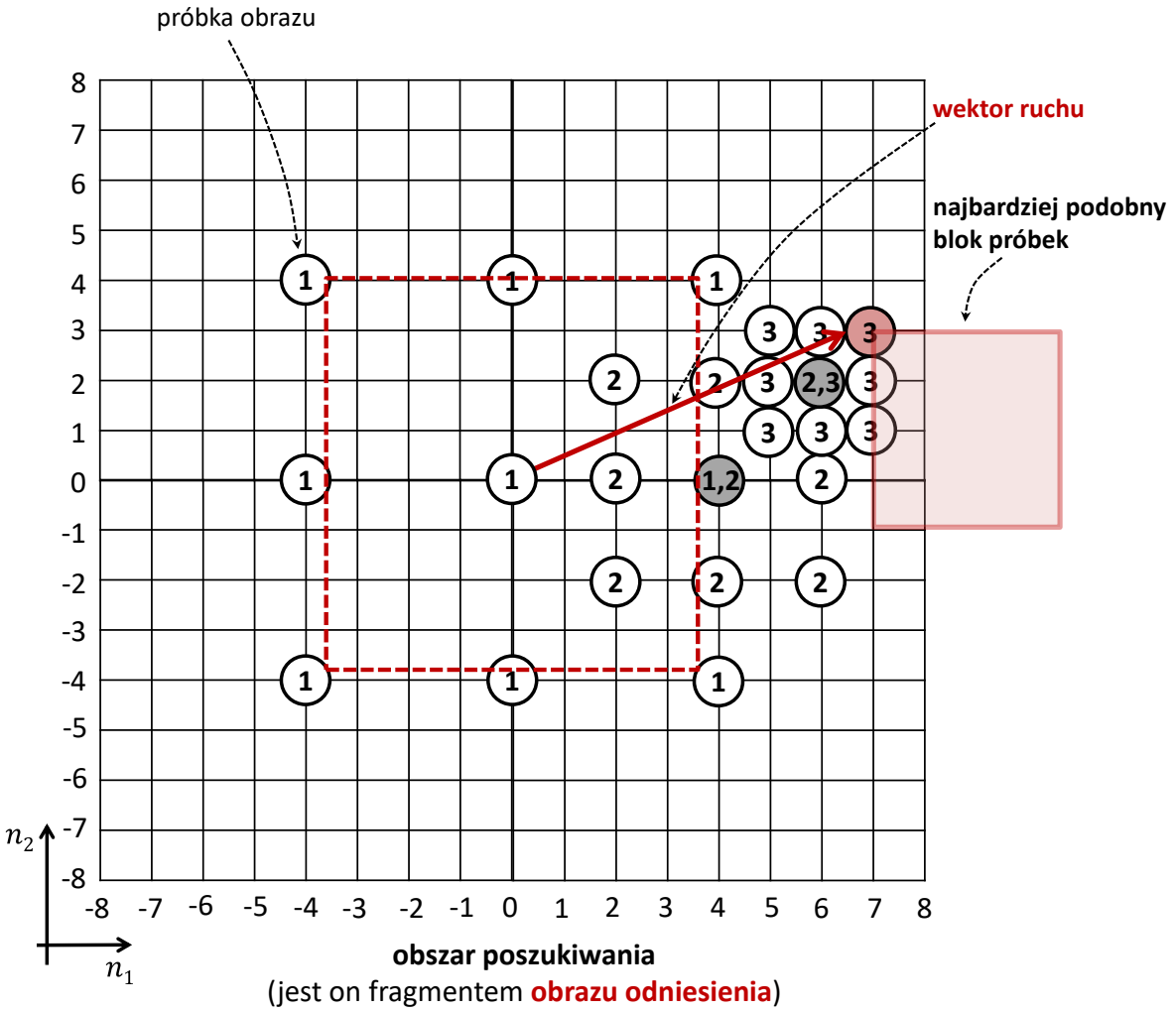
Dodatkowy komentarz: W przedstawionym na rysunku 4-59 przykładzie „najlepszy” blok z pierwszego rozmieszczenia punktów (punkty oznaczone numerem 1) związany był z „prawym” punktem znaku + (jego współrzędne (n_1, n_2) to $(4,0)$). A zatem wartość parametru s nie była na tym etapie redukowana. Nowo wybrane punkty zostały oznaczone numerem 2. Proszę zwrócić uwagę, że w tym konkretnym wyborze numerem 2 należałoby oznaczyć również punkt, którego współrzędne (n_1, n_2) wynoszą $(0,0)$. Nie zrobiono jednak tego celowo – skoro w poprzednim wyborze pięciu punktów punkt o współrzędnych $(0,0)$ nie dał „najlepszego” bloku, to nie ma sensu takiego punktu rozważać już w kolejnym wyborze punktów. Nic w ten sposób nie tracimy, a oszczędności w ilości wykonanych obliczeń są oczywiste.

Etap 3: Powtarzamy obliczenia **etapu 2** tak długo, aż rozmiar kroku przesunięcia s nie osiągnie wartości 1. W przypadku $s = 1$, począwszy od ostatnio uzyskanego punktu **centralnego** ustala się dziewięć finalnych punktów. Na rysunku 4-59 punkty te zostały oznaczone numerem 6. Jeden z tych dziewięciu punktów skojarzony jest z blokiem próbek obrazu odniesienia, który wedle „opinii” algorytmu wykazuje największe podobieństwo z aktualnie kodowanym blokiem.



Rysunek 4-59. Ilustracja działania metody **dwuwymiarowego poszukiwania logarytmicznego**. Jest to szybka metoda estymacji ruchu, która jest realizowana w ramach wcześniej ustalonego **obszaru poszukiwania**. Na rysunku przyjęto, że obszar poszukiwania ma wielkość 17x17 próbek obrazu.

Można powiedzieć, że bardzo podobnie do omówionego algorytmu działa druga z metod, czyli metoda **trzech kroków** (ang. three-step search). Zamiast jednak w każdym etapie obliczeń dokonywać porównania kodowanego bloku próbek z wybranymi, pięcioma blokami obrazu odniesienia robi się to w odniesieniu do dziewięciu bloków obrazu odniesienia, których rozmieszczenie jest takie jak na rysunku 4-60. W przypadku **algorytmu trzech kroków** parametr s podlega dwukrotnej redukcji po każdym etapie obliczeń. W przypadku założonego wcześniej rozmiaru obszaru, wewnątrz którego poszukiwano „najlepszego” bloku (był to rozmiar 17x17, bo $d = 8$) algorytm zakończy swoją pracę w trzech krokach, z czego wynika właśnie nazwa metody.

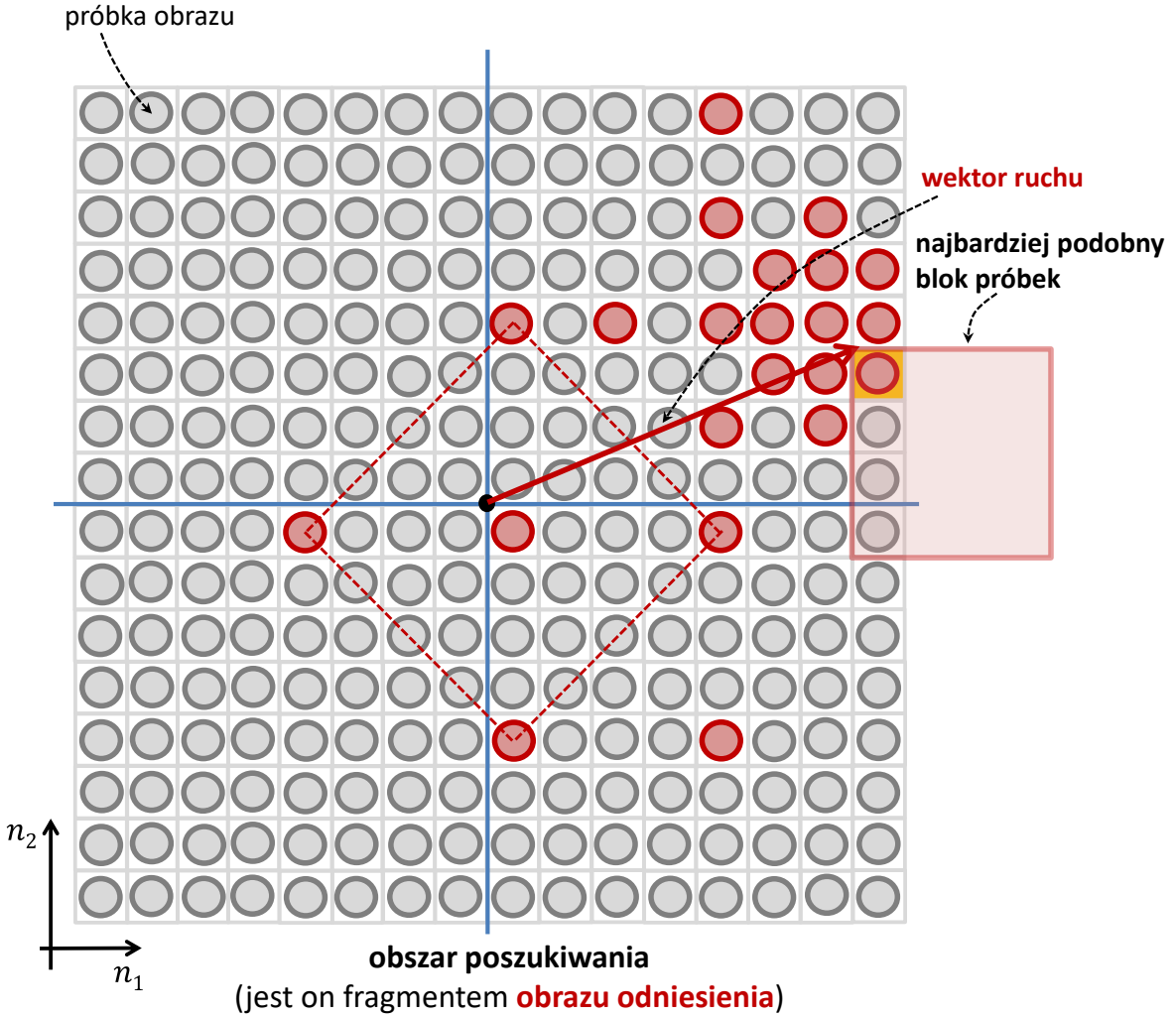


Rysunek 4-60. Ilustracja działania **algorytmu trzech kroków**. Jest to szybka metoda estymacji ruchu, która jest realizowana w ramach wcześniej ustalonego **obszaru poszukiwania**. Na rysunku przyjęto, że obszar poszukiwania ma wielkość 17x17 próbek obrazu.

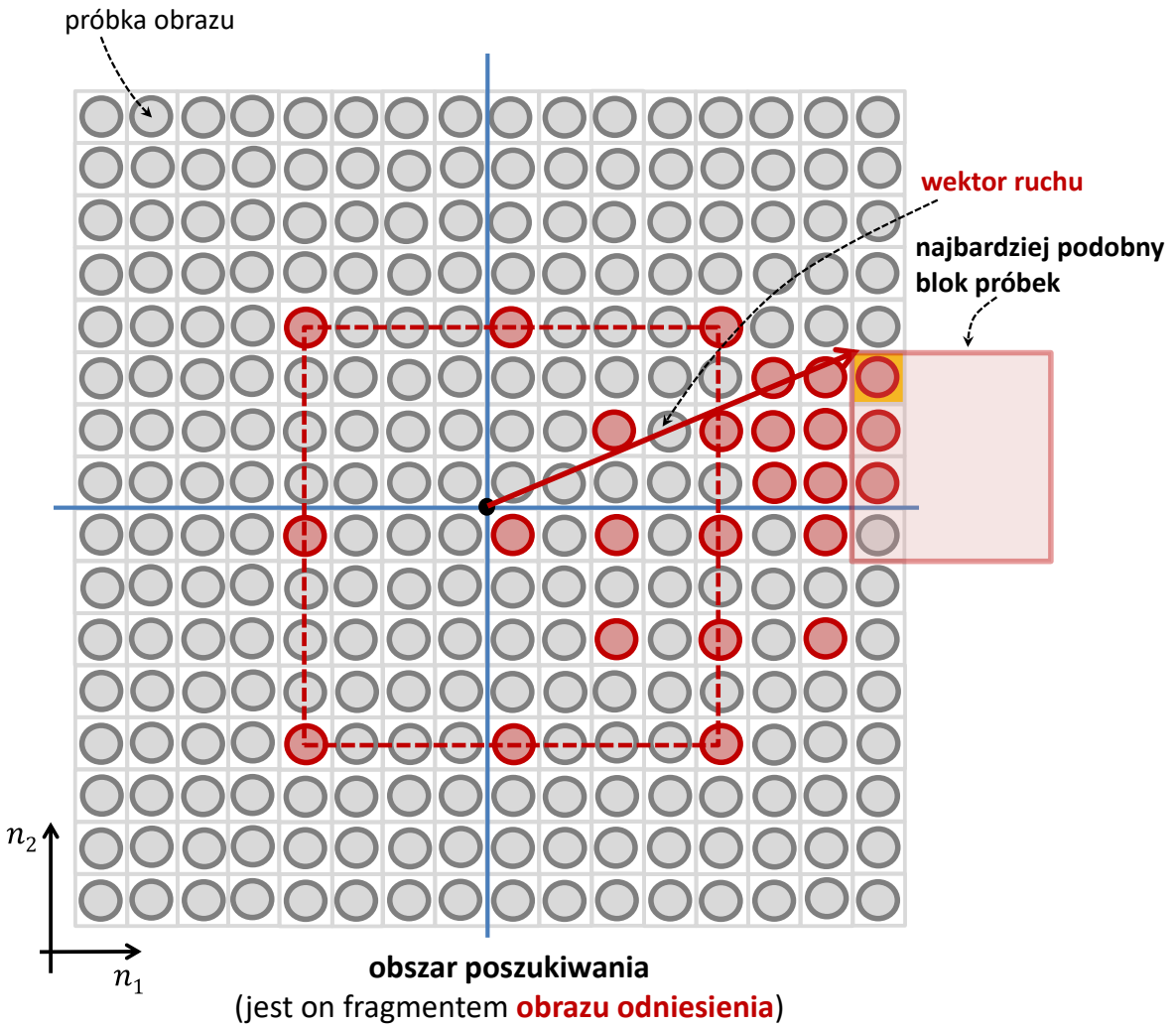
Zasada działania trzeciej z metod jest zgoła inna od dwóch pierwszych. Metoda **poszukiwania hierarchicznego** estymuje ruch wykonując odpowiednie obliczenia dla różnych wersji obrazów, które różnią się rozdzielczością przestrzenną. Wstępna estymacja ruchu jest wykonywana dla obrazów o zredukowanej rozdzielczości. Redukując rozdzielczość przestrzenną obrazów odpowiednio zmniejsza się również wielkość obszaru poszukiwania oraz rozmiar bloków, dla których wektory ruchu są wyznaczane. Otrzymane w ten sposób wektory ruchu uważa się za bardzo dobry punkt startowy dla dalszych prac, których celem jest uzyskanie wektorów dla obrazów o coraz wyższej rozdzielczości. W kolejnych krokach algorytmu zwiększa się więc rozdzielczość przestrzenną obrazów, dwukrotnie w pionie i w poziomie w każdym kroku, żeby bazując na poprzednich wynikach wyliczyć wektory ruchu dla nowych obrazów (czyli o zwiększonej rozdzielczości). Z uwagi na dwukrotne zwiększanie rozdzielczości obrazów w każdym kroku, względem punktu, na który wskazuje poprzednio otrzymany wektor ruchu, wystarczy zrealizować „doszukanie” w zakresie ±1 próbka w każdym kierunku w obrazie (poziomym i pionowym).

4.21. Szybkie metody estymacji ruchu – ocena skuteczności i złożoności obliczeniowej

Szybie metody estymacji ruchu drastycznie redukują liczbę porównywanych bloków obrazu. Już na pierwszy rzut oka można to bardzo łatwo zauważyć porównując ze sobą liczbę szarych i czerwonych punktów na rysunkach 4-61 i 4-62 (z danym punktem jest związany blok obrazu odniesienia, który podlega porównaniu z blokiem, który aktualnie kodujemy). Szybka metoda estymacji ruchu porównuje kodowany blok tylko z tymi blokami obrazu odniesienia, które są związane z „czerwonymi” punktami, podczas gdy tradycyjna metoda estymacji ruchu robi to w odniesieniu do bloków związanych z punktami tak czerwonymi jak i szarymi. Należy w tym miejscu dodatkowo dopowiedzieć, że różnica w liczbie wykonywanych porównań jest tym większa, im większy założy się rozmiar obszaru poszukiwania dla metody estymacji ruchu.



Rysunek 4-61. Ilustracja złożoności metody **dwuwymiarowego poszukiwania logarytmicznego**, w stosunku do metody **pełnego poszukiwania**. Metoda **dwuwymiarowego poszukiwania logarytmicznego** porównuje z blokiem aktualnie kodowanym jedynie te bloki obrazu odniesienia, które są skojarzone z próbkami oznaczonymi na rysunku kolorem czerwonym (przywołane próbki „czerwone” są lewą-górną próbką tych bloków). Metoda pełnego poszukiwania robi to w odniesieniu do wszystkich możliwych bloków obszaru poszukiwania, tzn. skojarzonych z próbkami zarówno „czerwonymi” jak i „szarymi”.



Rysunek 4-62. Ilustracja złożoności metody **trzech kroków**, w stosunku do metody **pełnego poszukiwania**. Metoda **trzech kroków** porównuje z blokiem aktualnie kodowanym jedynie te bloki obrazu odniesienia, które są skojarzone z próbkami oznaczonymi na rysunku kolorem czerwonym (przywołane próbki „czerwone” są lewą-górną próbką tych bloków). Metoda pełnego poszukiwania robi to w odniesieniu do wszystkich możliwych bloków obszaru poszukiwania, tzn. skojarzonych z próbkami zarówno „czerwonymi” jak i „szarymi”.

Dla przywołanych w tej książce metod estymacji ruchu można podać ponadto matematyczne wyrażenia, które dokładnie pokażą, jaka jest liczba operacji porównania dwóch próbek obrazu (jednej z bloku odniesienia i drugiej z bloku kodowanego), którą trzeba wykonać dla jednego bloku obrazu (o danym rozmiarze), znając przy tym wielkość obszaru poszukiwania. Na tej podstawie, uwzględniając dodatkowo przestrzenną rozdzielczość pojedynczego obrazu oraz liczbę obrazów w czasie 1 sekundy sekwencji, łatwo można policzyć, dla jakiej dokładnie liczby bloków wektory ruchu należy estymować (w czasie 1 sekundy). Ostatecznie daje to więc konieczną do wykonania liczbę operacji porównań dwóch próbek obrazu w czasie 1 sekundy. Zakładając, że estymujemy ruch dla sekwencji o parametrach podanych w tabeli 4.1, i wyznaczając wektory ruchu w blokach 16x16, przy założeniu obszaru poszukiwania o wielkości 65x65 (parametr $d = 32$), otrzymuje się następujące liczby operacji porównania próbek (patrz tabela 4.1). W założonym scenariuszu widzimy więc, że dla szybkiej metody poszukiwania hierarchicznego tych operacji jest aż **144-krotnie mniej** niż w przypadku zwykłej metody pełnego poszukiwania! Jest to więc

redukcja czasu wykonania estymacji ruchu aż o dwa rzędy wielkości! Przekłada się to oczywiście na bardzo znaczące skrócenie czasu kodowania sekwencji na skutek silnej redukcji złożoności kodera.

Tabela 4.1 Złożoność obliczeniowa wybranych metod estymacji ruchu.

Metoda	Liczba operacji porównania dwóch próbek dla jednego bloku **16x16** (**obszar poszukiwania:** (2**d**+1) x (2**d**+1))	Liczba operacji porównania na **1s** (**1920x1080@50Hz**) (**d=32**)
Pełne poszukiwanie	$(2\boldsymbol{d}+1)^2 \cdot \mathbf{16} \cdot \mathbf{16}$	$\mathbf{4{,}38 \cdot 10^{11}}$
Algorytm trzech kroków	$(8 \cdot \lceil log_2 \boldsymbol{d} \rceil + 1) \cdot 16 \cdot 16$	$\mathbf{4{,}25 \cdot 10^{9}}$
Poszukiwanie hierarchiczne	$\{(2 \cdot \lceil \boldsymbol{d}/4 \rceil + 1)^2 + 180\} \cdot 16 \cdot 16/4 \cdot 4$	$\mathbf{3{,}04 \cdot 10^{9}}$

$$\frac{\text{Pełne poszukiwanie: } 4{,}38 \cdot 10^{11}}{\text{Poszukiwanie hierarchiczne: } 3{,}04 \cdot 10^{9}} = 144\text{-krotna redukcja liczby operacji}$$

Z perspektywy ostatniej kwestii ciekawe jest porównanie rezultatów dwóch metod estymacji ruchu, **metody pełnego poszukiwania** i **szybkiej metody estymacji** znanej jako **TZ-search** [Purn12], które to metody działają w strukturze pełnego kodera wizyjnego nowej generacji. Tym koderem jest koder HEVC. Wyniki badań własnych autora pokazują, że zastosowanie w koderze HEVC algorytmu szybkiej estymacji ruchu, zamiast metody pełnego poszukiwania, skraca czas kodowania obrazów **od 14 do 25 razy** (zależnie od treści sekwencji oraz zastosowanej wartości parametru kwantyzacji QP), przy zmniejszeniu efektywności kompresji obrazów o **mniej niż 1%** (porównania efektywności dokonano miarą Bjøntegaarda [Bjønt01]). Takie wyniki zostały otrzymane dla modelowego oprogramowania HM 16.6 kodera HEVC. Tak dobre rezultaty są wystarczającym uzasadnieniem praktycznego wykorzystania szybkich metod estymacji ruchu w koderach ruchomego obrazu.

4.22. Międzyobrazowa predykcja treści obrazu. W jak dużych blokach najlepiej ją stosować?

Realizując międzyobrazową predykcję treści można się zastanawiać w jak dużych blokach obrazu najlepiej jest takie przewidywanie realizować? Mówiąc inaczej, czy sposób podziału obrazu na bloki ma jakiekolwiek przełożenie na uzyskiwane wyniki predykcji ich treści? Odpowiedź jest oczywiście twierdząca – ma i to bardzo duże.

Wiele reguł pozostaje tutaj takich samych jak miało to miejsce w przypadku predykcji wewnątrzobrazowej (patrz punkt 4.5). Predykcję treści najlepiej jest więc realizować w blokach o arbitralnie ustalanym przez koder rozmiarze, przy czym, koder powinien mieć do dyspozycji szeroki wachlarz tych rozmiarów. W zależności od treści aktualnie kodowanego fragmentu obrazu oraz stopnia podobieństwa tej treści z tym, co znajduje się w obrazie, który stanowi odniesienie dla

międzyobrazowej predykcji, koder powinien móc wybrać najlepszy wariant rozmiaru bloku, którego treść będzie starał się przewidzieć. Przechodząc na wyższy poziom powiemy, że koder powinien mieć możliwość ustalenia najlepszej siatki podziału kodowanego obrazu na bloki. Wyniki tego podziału silnie później rzutują na osiąganą efektywność kompresji danych. Dlatego najnowsze kodery obrazu, czyli takie, które zostały opracowane po roku 2010, mogą stosować bloki o rozmiarach od 4x4, poprzez 8x8 i 16x16, aż po bloki 32x32 i 64x64. Dodatkowo, możliwe są jeszcze dalsze podziały tych bloków (nie dotyczy to jednak bloku 4x4) na dwa mniejsze bloki prostokątne, stosując podział bloku w kierunku horyzontalnym lub wertykalnym, z podziałem na dwa bloki równe lub z podziałem niesymetrycznym.

Przykładowy, rzeczywisty rezultat podziału obrazu na bloki został przedstawiony na rysunku 4-63. Przykład ten prezentuje faktyczne decyzje, jakie podjął koder HEVC, działając w oparciu o modelowe oprogramowanie HM 10.0. Porównując ten rezultat z wynikami, jakie zostały wcześniej otrzymane dla przewidywania wewnątrzobrazowego (patrz zamieszczony w punkcie 4.5 rysunek 4-8) nie sposób nie zauważyć pewnej prawidłowości, zgodnie z którą, częściej zastosowanie mają w przedstawionym tutaj przykładzie duże bloki obrazu, czyli tym samym, spada użycie bloków o mniejszych rozmiarach. Nie ma tutaj tej prawidłowości, którą widać było w przypadku przewidywania wewnątrzobrazowego, że prosta treść obrazu przekłada się na użycie dużych bloków, a mocno złożona treść na podział fragmentu obrazu na wiele małych bloków. Tutaj tak nie jest. Wynika to naturalnie z samego mechanizmu międzyobrazowej predykcji treści – z obrazu z innej chwili czasowej „kopiujemy” podczas predykcji określony fragment obrazu, który bardzo przypomina ten, który aktualnie kodujemy. Przy tak działającej predykcji wysoki stopień podobieństwa dwóch bloków (kodowanego i odniesienia) często udaje się uzyskać również w przypadku większych bloków, bez konieczności ich podziału na mniejsze fragmenty (co z punktu widzenia odpowiedniej sygnalizacji o zastosowaniu takiego podziału byłoby kosztowne). Jest to więc jeden z elementów, który decyduje o wyższości predykcji międzyobrazowej nad wewnątrzobrazową.

Rysunek 4-63. Obraz testowy wraz z przykładowym podziałem obrazu na bloki o określonym rozmiarze. Otrzymana siatka bloków jest wynikiem podjętych przez modelowe oprogramowanie HM 10.0 kodera HEVC decyzji odnośnie do rozmiarów bloków, w których najlepiej jest zastosować predykcję międzyobrazową treści, wraz z dalszym kodowaniem danych. Przykład został opracowany z użyciem oprogramowania analizatora HEVC, którego głównym wykonawcą jest Pan Piesik Daniel.

4.23. Jeszcze o praktycznej realizacji estymacji ruchu – bardzo krótkie podsumowanie

Celem uzyskania najlepszych proporcji pomiędzy złożonością algorytmu estymacji ruchu, a wynikami efektywności kompresji danych w praktyce zastosowanie mają następujące rozwiązania:

1) Kodery korzystają z szybkich metod estymacji ruchu. Potrafią one skrócić czas działania kodera obrazu około **20-krotnie**. Dlatego metody pełnego poszukiwania w zasadzie nie są przedmiotem praktycznego wykorzystania.

2) Realizowana uproszczonym algorytmem estymacja ruchu jest wykonywana w ramach obszaru poszukiwania, który został z góry zawężony. Dość częstą praktyką są obszary o wielkości 33x33 bądź 65x65 próbek obrazu.

3) Ruch estymuje się w blokach o arbitralnie ustalanej przez koder wielkości. W najnowszych rozwiązaniach są to bloki o wielkości z zakresu od 4x4 do 64x64.

4) Estymację ruchu realizuje się tylko dla składowej luminancji Y obrazu. Z uwagi na mniejszą istotność składowych koloru C_B i C_R dla percepcji obrazu, podczas kodowania składowych koloru korzysta się z wyników estymacji ruchu otrzymanych dla składowej luminancji.

Przytoczone powyżej warunki estymowania ruchu w sekwencjach wizyjnych prowadzą do silnej redukcji złożoności algorytmu. Jednak pomimo tego udział estymacji ruchu jest w koderze obrazu ciągle wysoki – stanowi ona **30% - 70%** wszystkich obliczeń współczesnego kodera (np. HEVC).

4.24. Estymacja ruchu metodą pasowania bloków. Jak ocenić stopień podobieństwa dwóch bloków obrazu?

Wyszukanie w obrazie odniesienia najbardziej podobnego bloku do tego, który aktualnie kodujemy wymaga dokonania oceny stopnia ich podobieństwa. Taką ocenę łatwo może zrobić człowiek, oglądając treść bloków, jednak z perspektywy programu komputerowego sposób tej oceny nie jest już taki oczywisty.

Uwzględniając otrzymywane rezultaty stopnia podobieństwa dwóch bloków, jak również obliczeniową złożoność takiego porównania, najczęściej stosuje się w koderze obrazu jedną z trzech poniższych miar podobieństwa:

1) **Sumę bezwzględnych różnić** (ang. sum of absolute differences – **SAD**) odpowiadających sobie wartości próbek w dwóch porównywanych blokach. Matematycznie miarę tę określa się następująco:

$$SAD(x,y;\boldsymbol{d_1},\boldsymbol{d_2}) = \sum_{l_1=0}^{N_1-1}\sum_{l_2=0}^{N_2-1}\left|f_{blok\ kodowany}(x+l_1,y+l_2) - f_{blok\ odniesienia}(x+l_1+\boldsymbol{d_1},y+l_2+\boldsymbol{d_2})\right| \tag{4.2}$$

gdzie:
(x, y) – współrzędne (w obrazie aktualnie kodowanym) lewego górnego narożnika bloku próbek, dla którego poszukuje się najbardziej podobnego bloku próbek w obrazie odniesienia;

$f_{blok\ kodowany}(x+l_1, y+l_2)$ – funkcja opisująca wartości próbek aktualnie kodowanego bloku (liczby w nawiasie określają położenie próbek wewnątrz obrazu);

$f_{blok\ odniesienia}(x+l_1+\boldsymbol{d_1}, y+l_2+\boldsymbol{d_2})$ – funkcja opisująca wartości próbek bloku z obrazu odniesienia (liczby w nawiasie określają położenie próbek wewnątrz obrazu);

$[\boldsymbol{d_1}, \boldsymbol{d_2}]$ – składowe poszukiwanego wektora ruchu;

$\boldsymbol{N_1}, \boldsymbol{N_2}$ – rozmiar (w próbkach) bloku próbek.

2) **Sumę kwadratów różnić** (ang. sum of squared differences – **SSD**) odpowiadających sobie wartości próbek w dwóch porównywanych blokach. Wzór matematyczny dla tej miary jest następujący:

$$SSD(x,y;\boldsymbol{d_1},\boldsymbol{d_2}) = \sum_{l_1=0}^{N_1-1}\sum_{l_2=0}^{N_2-1}\left(f_{blok\ kodowany}(x+l_1,y+l_2) - f_{blok\ odniesienia}(x+l_1+\boldsymbol{d_1},y+l_2+\boldsymbol{d_2})\right)^2 \tag{4.3}$$

Znaczenie przyjętych w tym wzorze oznaczeń jest takie jak w przypadku miary SAD.

3) **Sumę bezwzględnych różnić w dziedzinie transformaty** (ang. sum of absolute transformed differences – **SATD**) odpowiadających sobie wartości próbek transformaty wyznaczonych dla próbek obrazu w dwóch porównywanych blokach. Tę miarę określa się równaniem:

$$SATD(x, y; \boldsymbol{d_1}, \boldsymbol{d_2}) = \sum_{l_1=0}^{N_1-1} \sum_{l_2=0}^{N_2-1} \left| F_{blok\ kodowany}(x + l_1, y + l_2) - F_{blok\ odniesienia}(x + l_1 + \boldsymbol{d_1}, y + l_2 + \boldsymbol{d_2}) \right| \quad (4.4)$$

gdzie:
$F_{blok\ kodowany}(x + l_1, y + l_2)$ – funkcja opisująca wartości próbek transformaty aktualnie kodowanego bloku próbek (liczby w nawiasie określają położenie próbek transformaty wewnątrz obrazu);

$F_{blok\ odniesienia}(x + l_1 + \boldsymbol{d_1}, y + l_2 + \boldsymbol{d_2})$ – funkcja opisująca wartości próbek transformaty bloku z obrazu odniesienia (liczby w nawiasie określają położenie próbek transformaty wewnątrz obrazu);

Pozostałe oznaczenia mają takie znaczenie jak w przypadku miary **SAD**.

W przypadku każdej z tych miar im mniejsza jest jej wartość, tym większe podobieństwo dwóch porównywanych bloków. W skrajnym przypadku, wartość miary równa 0 oznacza, że porównywane bloki są dokładnie takie same. W związku z powyższym, zamiast o miarach podobieństwa, należałoby w tym punkcie raczej mówić o miarach odmienności bloków.

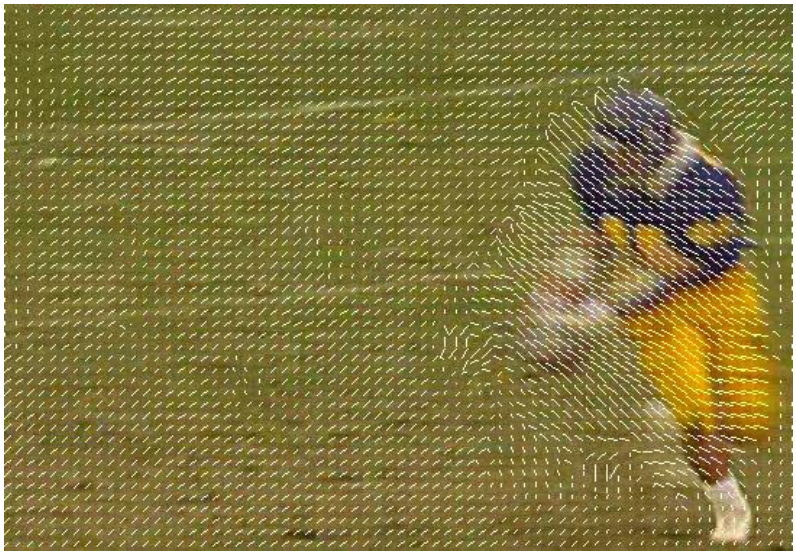
Spośród wskazanych powyżej miar najbardziej obliczeniowo złożona jest miara **SATD**. Operuje ona w dziedzinie transformaty sygnału, dlatego przed jej zastosowaniem konieczna jest transformacja próbek dwóch porównywanych bloków obrazu, z dziedziny przestrzennej do dziedziny częstotliwości. Oczywiście wykonanie tej transformacji wiąże się z określonymi, dodatkowymi nakładami obliczeniowymi, co zauważalnie wydłuża procedurę porównania bloków. Dlatego na potrzeby samej miary, zamiast przekształcenia **DCT** stosuje się zwykle prostszą obliczeniowo transformację **Walsha-Hadamarda**. Badania własne autora wskazują jednak, że pomimo tego uproszczenia stosowanie miary **SATD** i tak zwiększa złożoność metody porównania bloków aż **4,5-krotnie** (w przypadku czystej implementacji w C++), i około **3-krotnie** (w przypadku skorzystania w kodzie programu z instrukcji AVX procesora), w porównaniu z miarami, które nie stosują przejścia do dziedziny częstotliwości. Większe nakłady obliczeniowe w metryce **SATD** owocują większą efektywnością kompresji danych.

Porównując ze sobą miary **SAD** i **SSD**, obliczeniowo najprostsza jest ta pierwsza. Tak jest przynajmniej w przypadku implementacji C++ wymienionych miar, kiedy to użycie **SSD** zwiększa złożoność estymacji ruchu około **1,5 raza**. Jednak, co ciekawe, w przypadku implementacji metody estymacji ruchu z wykorzystaniem multimedialnych rozszerzeń AVX procesora Intel różnicy w złożoności algorytmu już nie będzie. Praktyka pokazuje, że z pomocą **SSD** osiągniemy lepsze efekty kompresji niż przy wykorzystaniu **SAD**.

Przytoczone powyżej wyniki analizy złożoności poszczególnych miar zostały otrzymane implementując miary w algorytmie TZ Search estymacji ruchu, w koderze HEVC. Obliczenia były prowadzone na procesorze Intel Core i7 – 5820K (3,6 GHz), przy 6 aktywnych wątkach obliczeniowych. Bardziej szczegółowe dane można znaleźć w zespołowej pracy autora [Stan16].

4.25. Estymacja ruchu metodą pasowania bloków. Czy ta metoda trafnie wyznacza ruch obiektów sekwencji?

Chociaż w praktyce estymacja ruchu realizowana jest najczęściej z użyciem **metody pasowania bloków**, to jak się bardzo często okazuje metoda ta wcale nie określa rzeczywistego ruchu obiektów sceny. Wynikowe wektory ruchu są w przypadku tej metody często niezgodne z faktycznym kierunkiem przemieszczania się w czasie poszczególnych fragmentów obiektów sceny. Można to zaobserwować porównując ze sobą dwa różne rezultaty estymacji ruchu: pierwszy, otrzymany metodą **przepływu optycznego** (ang. optical flow) [Doma10], która daje w wyniku pole ruchu w dużym stopniu zgodne z faktycznym kierunkiem ruchu obiektów w scenie, oraz drugi, otrzymany metodą **pasowania bloków** (patrz rysunek 4-64).



wektory ruchu – metoda **przepływu optycznego**

wektory ruchu – metoda **pasowania bloków**

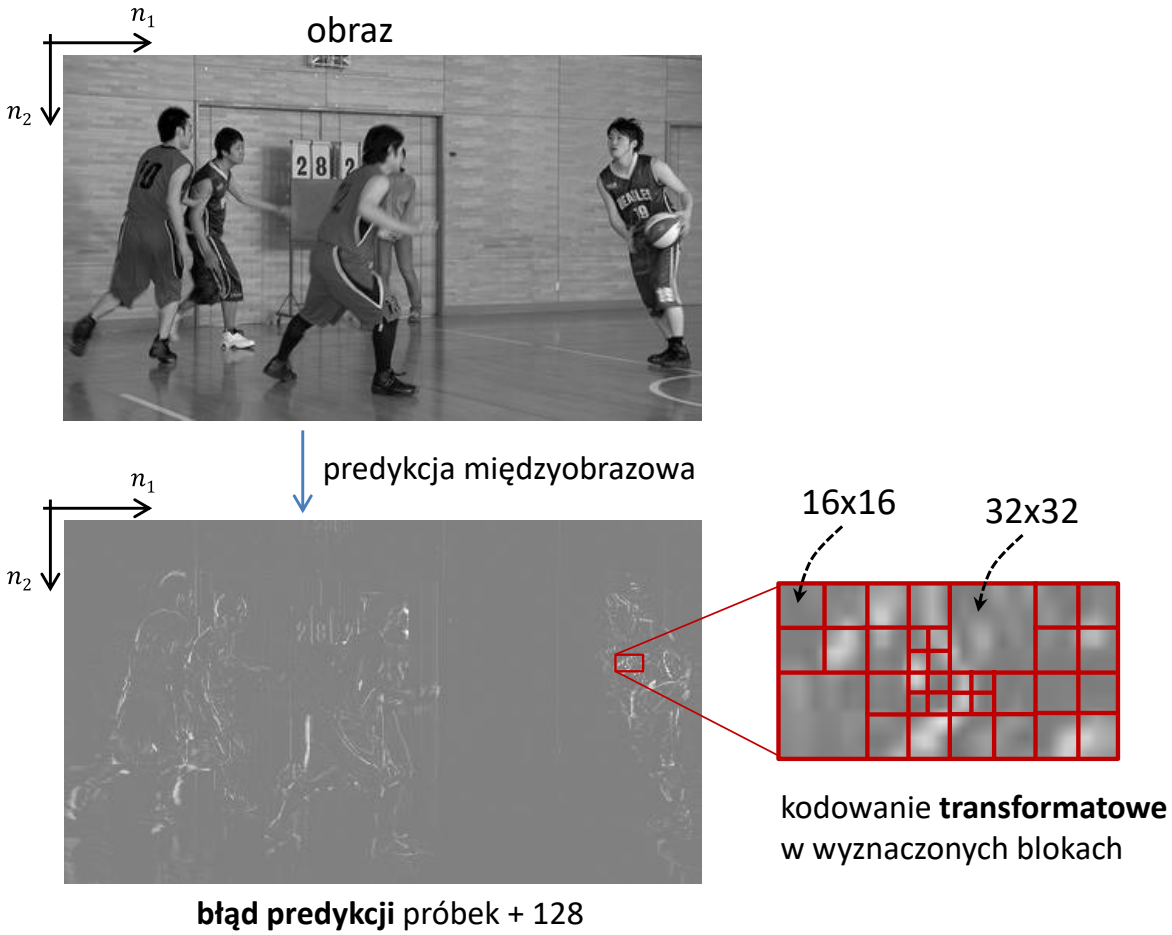
Rysunek 4-64. Przykładowe wyniki estymacji ruchu (są nimi wektory ruchu oznaczone na obrazach białymi kreskami), które otrzymano stosując dwie różne metody estymacji: **przepływu optycznego** i **pasowania bloków**.

Uzasadnienia przedstawionej na rysunkach rozbieżności wyników, które zostały otrzymane dla wskazanych metod estymacji ruchu, należy szukać w samym kryterium podobieństwa dwóch bloków (bloku kodowanego i bloku z obrazu odniesienia), którym podczas wyznaczania wektorów ruchu posługuje się **metoda pasowania bloków**. W tej metodzie wektor ruchu wskazuje na położenie w obrazie odniesienia bloku, którego treść jest w największym stopniu podobna do treści bloku, który aktualnie kodujemy. Nie ma więc tutaj bezpośredniego odniesienia do rzeczywistego ruchu obiektów sceny. Z uwagi na występujące w obrazie sygnały szumu, jak również niektóre zmiany treści obrazów, które nie stanowią założonego, prostego ruchu translacyjnego obiektów na płaszczyźnie obrazu, ten najbardziej podobny blok obrazu odniesienia może mieć zupełnie inne położenie niż wynika to z faktycznego ruchu obiektów w scenie. Dlatego w metodzie pasowania bloków wektory ruchu wyznaczone dla sąsiednich bloków obrazu mogą się znacznie między sobą różnić. Wynikowe pole ruchu może być bardzo niespójne, wręcz chaotyczne, co dobrze widać w przedstawionym przykładzie. Praktyka pokazuje jednak, że rezultaty metody pasowania bloków dają i tak najlepsze wyniki kompresji obrazów (pole ruchu jest z reguły niespójne, ale za to błąd przewidywania treści bloków jest bardzo mały).

4.26. Kodowanie transformatowe błędu przewidywania treści bloków

Dokładnie tak jak miało to miejsce w przypadku kodowania z wewnątrzobrazową predykcją treści, błąd przewidywania międzyobrazowego jest w dalszej kolejności przedmiotem **stratnego kodowania transformatowego** (patrz idea przedstawiona na rysunku 4-65). Najkrócej rzecz ujmując, sygnał błędu predykcji próbek bloku o zadanej wielkości opisuje się „kolekcją” funkcji typu kosinusoidalnego o danej częstotliwości. W poszczególnych blokach amplitudy otrzymanych składowych harmonicznych poddaje się kwantyzacji (w celu określenia dokładności ich reprezentacji w zakodowanym strumieniu bitowym, przy uwzględnieniu zdolności percepcji poszczególnych składowych przez człowieka), a otrzymany wynik koduje się entropijnie.

Najważniejsze zasady kodowania próbek błędu są więc takie jak w przypadku kompresji wewnątrzobrazowej. Dlatego, w celu poznania większej ilości szczegółów, autor odsyła czytelnika do treści punktu 4.10.2.2.



Rysunek 4-65. Kodowanie transformatowe próbek błędu predykcji międzyobrazowej. Kodowanie jest realizowane w blokach, nazywanych blokami transformaty.

4.27. Kompletny koder obrazu, czyli łączne użycie wielu narzędzi kodowania danych

Koder ruchomego obrazu korzysta ze wszystkich narzędzi kompresji danych, jakie zostały do tej pory przedstawione w książce. Schematy blokowe kodera i dekodera obrazu wyglądają więc tak jak na zamieszczonych poniżej rysunkach 4-66 i 4-67.

Patrząc na pierwszy schemat, działanie kodera daje się opisać w kilku następujących zdaniach. Przed właściwą kompresją obraz dzielony jest na bloki o ustalonym rozmiarze, przy czym każdy z bloków może mieć ten rozmiar inny. Kompresji podlegają więc kolejne bloki obrazu. Jednak koder nie koduje bezpośrednio treści każdego z bloków. Stara się najpierw tę treść przewidzieć (dokonujemy predykcji treści bloku) na podstawie innych danych, które mają w swojej dyspozycji zarówno koder, jak i dekoder. Są to więc dane, które pochodzą z aktualnie kodowanego obrazu – mówimy w takim przypadku o **predykcji wewnątrzobrazowej** lub dane te znajdują się w obrazach z innych chwil czasu – mowa o **predykcji międzyobrazowej**. Żadna z tych predykcji nie jest jednak idealna, w ich toku jest popełniany pewien błąd. Jednak zwykle jest on bardzo mały. Błąd ten, nazywany **sygnałem błędu predykcji próbek** bloku, wyznacza się porównując aktualnie kodowany blok obrazu z tym, co wyszło z samego przewidywania. W koderze obrazu sygnał błędu predykcji próbek bloku jest przedmiotem dalszej kompresji, stosując kombinację stratnego

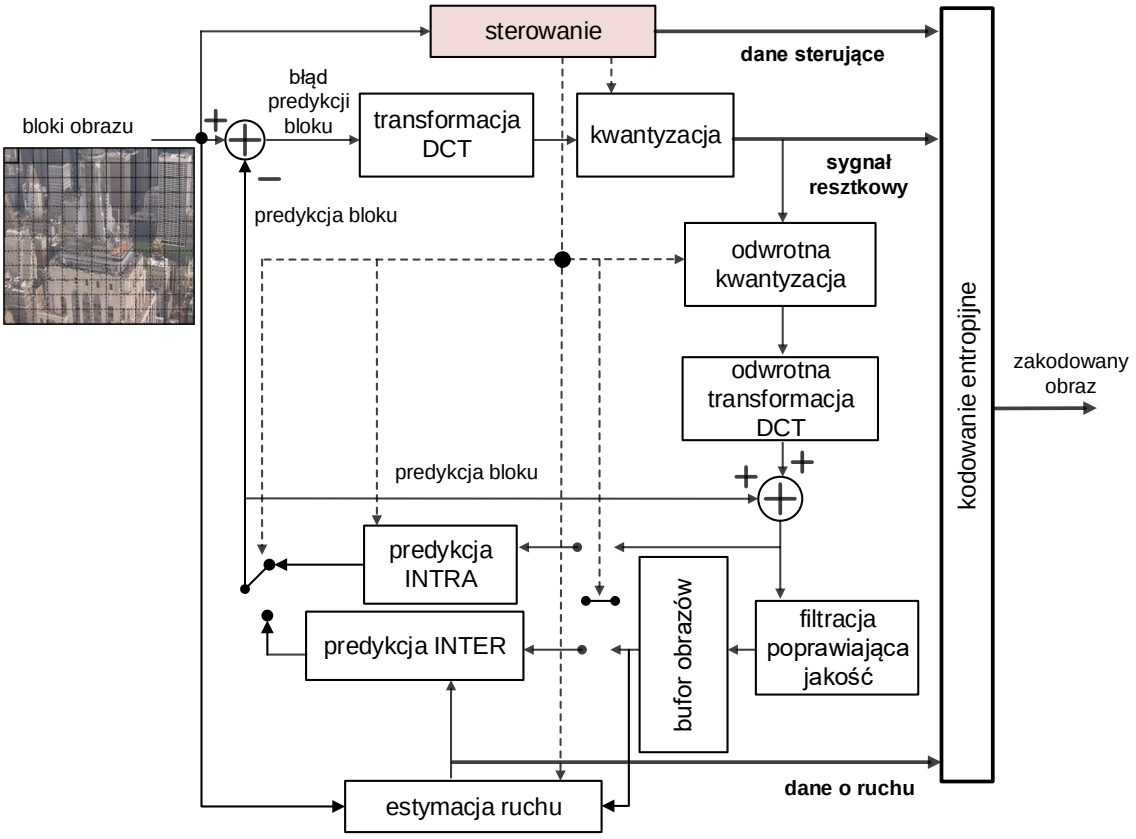
kodowania transformatowego oraz kodowania entropijnego, które jest już bezstratne. W przypadku każdego bloku obrazu koder wysyła więc do dekodera trzy rodzaje danych:

1) **Dane sterujące**, które informują dekoder o zastosowanym trybie (sposobie) kodowania bloku. Jest to więc informacja o trybie kompresji bloku (kodowanie wewnątrzobrazowe, kodowanie międzyobrazowe), dane na temat wielkości bloku, rodzaju zastosowanego predyktora treści bloku, w przypadku kompresji wewnątrzobrazowej, itd.

2) **Informacja o ruchu**, w przypadku zastosowania kodowania międzyobrazowego. Są to wektory ruchu i indeksy obrazów odniesienia.

3) **Błąd predykcji próbek** bloku w dziedzinie transformaty.

Powtórzmy to raz jeszcze, przedstawiony schemat działania kodera należy powtórzyć dla kolejnych bloków kodowanego obrazu.

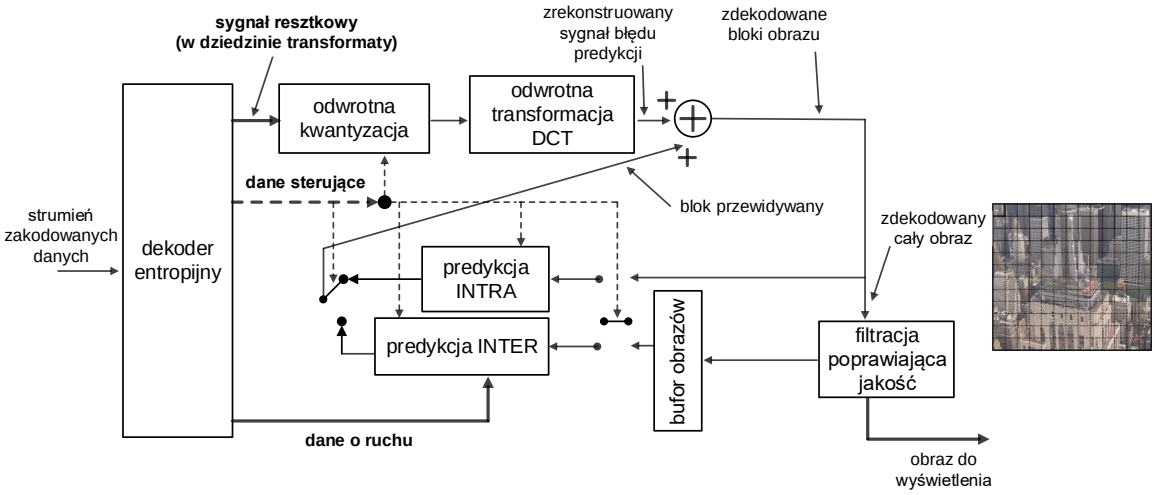


Rysunek 4-66. Schemat blokowy **hybrydowego kodera** ruchomego obrazu.

Z idei działania kodera wynika sposób działania **dekodera**. Określone operacje są w dekoderze wykonywane w odwrotnej niż w koderze, kolejności, również na poziomie bloków obrazu. Dekoder otrzymuje na swoim wejściu strumień zakodowanych danych. Pierwszym etapem dekompresji jest więc dekodowanie entropijne zakodowanego strumienia danych. W ten sposób dekoder „dociera” do trzech strumieni danych: 1) dane sterujące, 2) dane o ruchu, oraz 3) dane o błędzie predykcji próbek obrazu. Ponieważ wcześniej w koderze błąd predykcji próbek bloku

został poddany transformacji (zwykle DCT), a wynikowe współczynniki transformaty dodatkowo skwantowano, trzeci z wymienionych typów danych należy w dekoderze poddać operacjom dekwantowania[44] oraz odwrotnej transformacji. Wynikiem tych czynności jest zrekonstruowany sygnał błędu predykcji próbek, zrekonstruowany, ponieważ na skutek przeprowadzonej w koderze kompresji stratnej sygnał ten nie jest identyczny z sygnałem błędu predykcji próbek bloku, który był kodowany po stronie kodera. Do tak otrzymanego sygnału błędu predykcji próbek należy oczywiście dodać sam wynik przewidywania treści bloku. Koder dokładnie instruuje dekoder jak ma to przewidywanie zrobić, czy w drodze predykcji wewnątrzobrazowej (INTRA), czy międzyobrazowej (INTER), przy założeniu jakiego rozmiaru bloku obrazu, oraz z użyciem jakich dokładnie danych odniesienia (w przypadku predykcji INTER, który z obrazów z bufora obrazów jest odniesieniem dla predykcji treści bloku, w przypadku predykcji INTRA, próbki którego z bloków, które sąsiadują z kodowanym blokiem, mają być odniesieniem dla predykcji). Dodając rezultat przewidywania treści bloku do zrekonstruowanego sygnału błędu predykcji tej treści, otrzymuje się zdekodowaną wersję bloku obrazu.

Powyższe czynności dekodera należy wykonać dla kolejnych bloków obrazu.



Rysunek 4-67. Schemat blokowy **hybrydowego dekodera** zakodowanej sekwencji.

Kodowanie stratne obrazu wprowadza do obrazów szereg zniekształceń, które pogarszają jego jakość. Najważniejsze rodzaje takich zniekształceń zostały omówione w punktach 4.10.2.4.1 i 4.10.2.4.2. Kodery obrazu starszych generacji (np. koder MPEG-2) nie podejmowały żadnych czynności naprawczych. Inaczej jest w przypadku nowszych koderów obrazu, czyli takich, które zostały opracowane po roku 2000. Ponieważ zniekształcenia kompresji stratnej obniżają zarówno subiektywną jakość obrazów na wyjściu dekodera, jak i efektywność kompresji obrazów (o czym powiedziano szczegółowo w punktach 4.10.2.4.1 i 4.10.2.4.2) to na wyjściu nowszych koderów i dekoderów stosuje się pewne filtracje, które na rysunkach 4-66 i 4-67 zostały ujęte w bloku funkcjonalnym „filtracja poprawiająca jakość”. Wspomnianej filtracji poddaje się próbki zdekodowanego obrazu, przed jego ostatecznym wyświetleniem na ekranie. Jednak robi się to jeszcze w koderze i dekoderze obrazu.

[44] Chociaż w praktyce takie określenie się stosuje to trzeba mieć świadomość, że nie da się całkowicie odwrócić skutków kwantowania danych. Przywołane „dekwantowanie” przywraca jedynie pierwotny zakres dynamiczny danych, do stanu sprzed operacji kwantowania.

4.28. Jeszcze o filtracji, która poprawia jakość obrazów

Filtracje obrazu, które redukują efekt blokowy, jak również zniekształcenia treści w obrębie krawędzi obiektów istotnie poprawiają postrzeganą przez widza jakość obrazu po dekompresji. Dlatego jesteśmy zainteresowani tym, żeby wspomniane filtracje faktycznie stosować.

Jednak w tym miejscu można się zastanawiać, dlaczego te filtracje przeprowadza się w samym kodeku obrazu, czyli w koderze i dekoderze, a nie poza ich obrębem? Czy nie lepiej byłoby odciążyć koder i dekoder z tego niełatwego przecież zadania i poprawiać jakość obrazu w innym miejscu, np. w urządzeniu wyświetlacza, przed właściwą prezentacją zdekodowanego obrazu użytkownikowi? Odpowiedź jest tutaj następująca: ostatnie rozwiązanie, chociaż również poprawiałoby jakość obrazów, to skutkowałoby jednak gorszymi wynikami kompresji danych.

Żeby lepiej zrozumieć motywację przyjętego rozwiązania, wystarczy porównać ze sobą trzy obrazy (patrz rysunek 4-68):

1) obraz oryginalny, który będzie podlegał kompresji;

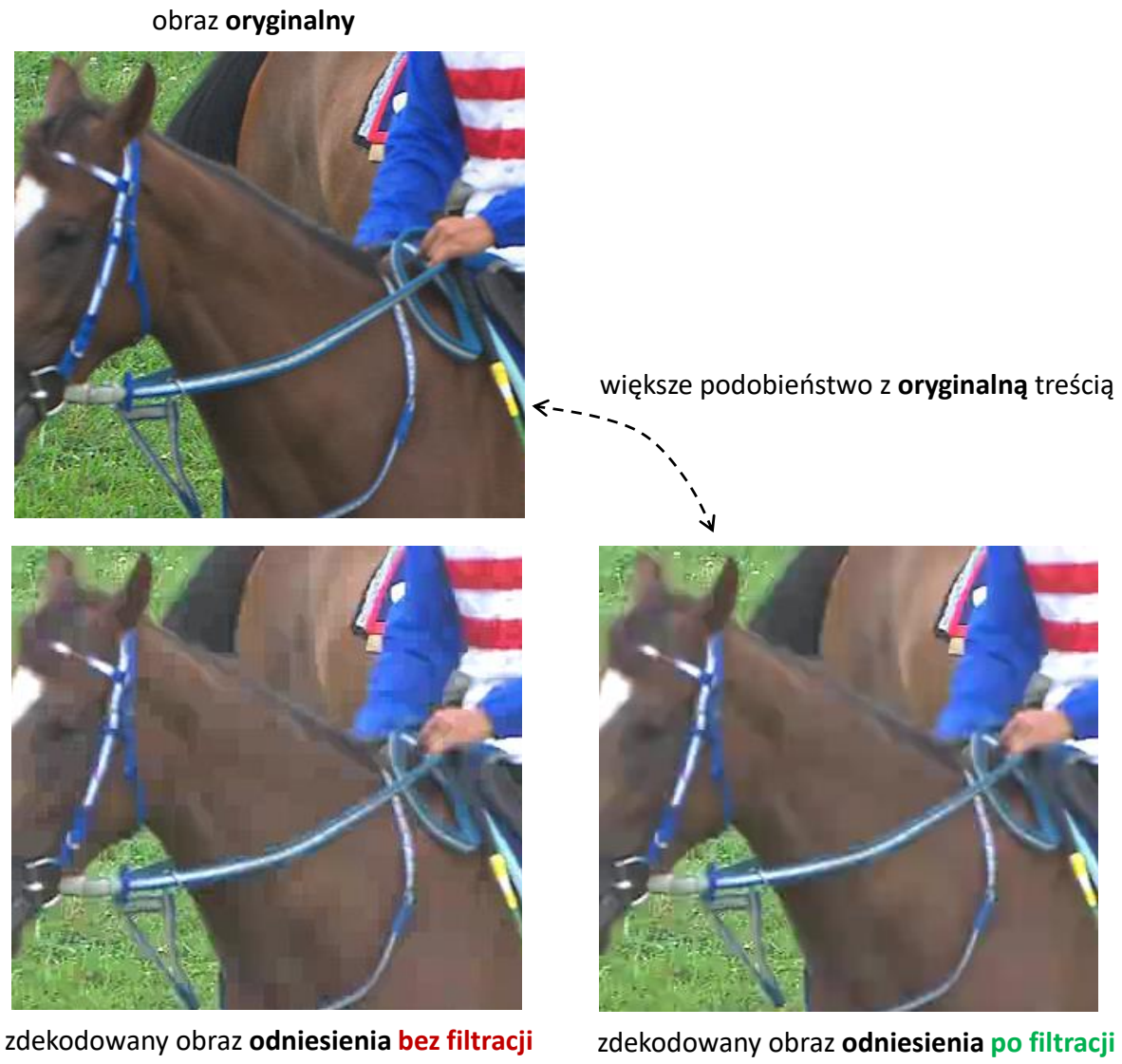
2) obraz odniesienia po kompresji i dekompresji, w którym występują błędy kodowania typowe dla efektów blokowego i dzwonienia (tętnień);

3) obraz odniesienia po kompresji i dekompresji, w którym zredukowane wspomniane efekty.

Zdekodowane obrazy odniesienia nie są oczywiście takie same jak aktualnie kodowany obraz oryginalny, i to bez względu na to, czy obrazy odniesienia były poddane przedmiotowej filtracji, czy też nie. Jednak wyraźnie widać, że z dwójki obrazów odniesienia ten, w którym filtracja była zastosowana wykazuje większe podobieństwo z treścią kodowanego obrazu. W końcu, w toku filtracji, z obrazu odniesienia zostały usunięte (czy raczej zredukowane) zniekształcenia, które nie występują przecież w treści aktualnie kodowanego obrazu. Kodując więc fragment bieżącego obrazu, lepiej jest przewidywać jego treść na podstawie takiej wersji obrazu odniesienia, w którym wcześniej zastosowana została filtracja, która poprawia jego jakość. Przy takim rozwiązaniu przewidywanie treści będzie trafniejsze, dokładniejsze. Mniejszy więc będzie błąd predykcji treści, co przełoży się na wyższą efektywność kompresji danych. Oprócz poprawy jakości samych obrazów, uzyskujemy więc dodatkowo poprawę wyników kompresji danych.

Żeby koder i dekoder mogły skorzystać z tego „dobrodziejstwa”, filtrację poprawiającą jakość należy umieścić w samym kodeku. Ulokowanie filtracji ma miejsce w pętli sprzężenia kodera i dekodera (patrz rysunki 4-66 i 4-67). W ten sposób, do bufora obrazów (do którego w trakcie kodowania bieżącego obrazu odwołują się koder i dekoder) trafiają zdekodowane obrazy, po przeprowadzeniu wspomnianych filtracji.

Opisany sposób przewidywania treści jest we współczesnych koderach (np. AVC, HEVC) stosowany powszechnie w przypadku międzyobrazowego kodowania obrazów sekwencji wizyjnej (patrz dalsza część książki).



Rysunek 4-68. Wzajemne porównanie trzech obrazów: oryginalnego, obrazu odniesienia po dekompresji i zastosowaniu filtracji poprawiającej jakość, oraz obrazu odniesienia po dekompresji, jednak bez zastosowania omawianej filtracji.

4.29. Pętla sprzężenia zwrotnego w koderze obrazu

W przedstawionym schemacie kodera obrazu najbardziej zastanawiająca może być obecność bloków odwrotnej kwantyzacji sygnału resztkowego, odwrotnej transformacji DCT, oraz operacji dodania do tak otrzymanego wyniku sygnału predykcji treści bloku. Czyli operacji, które realizują dekodowanie dopiero co zakodowanej treści bloku obrazu. Jaki jest zatem sens umieszczenia w koderze bloków funkcjonalnych dekodowania danych? Czy obecność tych bloków w koderze nie jest nadmiarowa? Te kwestie były już przedmiotem szczegółowych rozważań, kiedy mowa była o kompresji z predykcją wewnątrzobrazową (patrz punkt 4.10.2.1). Jako, że sformułowane tam konkluzje dotyczą również kompresji z predykcją międzyobrazową to dla zachowania spójności opisu zostaną one ponownie przytoczone w tym miejscu.

Przewidywanie próbek bloków musi się odbywać na bazie odtworzonych (czyli zdekodowanych), a nie oryginalnych próbek, które stanowią odniesienia dla predykcji treści. I to również po stronie kodera obrazu. Dla każdego z bloków obrazu koder dokonuje przewidywania ich treści i przesyła do dekodera informację o tym, w jakim zakresie treści kodowanego bloku nie udało się poprawnie przewidzieć na podstawie próbek odniesienia. Czyli koder wysyła do dekodera informację o tym, jaki jest błąd predykcji próbek bloku. Ta informacja jest bardzo potrzebna dekoderowi obrazu – błąd predykcji należy dodać do wyniku predykcji treści bloku, żeby dokonać właściwego jego odtworzenia (zdekodowania). Żeby zapewnić zatem pełną zgodność przebiegu procesów przewidywania treści w koderze i dekoderze obrazu, nie może być ona (predykcja) realizowane na podstawie oryginalnych próbek odniesienia. Takimi próbkami nie dysponuje dekoder obrazu, w przypadku stratnej jego kompresji. Sytuacja, w której koder stosuje jako odniesienie oryginalną wersję próbek (proszę zauważyć, że koder ma do nich dostęp), a dekoder do tego samego celu używa próbek o innych już wartościach (bo są wynikiem stratnej kompresji i dekompresji) prowadziłaby do **dryfu kodowania**. Pomiędzy stroną kodującą i dekodującą pojawiłaby się zatem rozbieżność rezultatów predykcji, która z uwagi na predykcyjne kodowanie danych szybko by się pogłębiała wraz z przetwarzaniem kolejnych bloków obrazu.

W związku z powyższym faktem, koder, przewidując treść bloków musi bazować na takiej wersji próbek odniesienia, którymi będzie dysponował dekoder obrazu. Stąd właśnie obecność na schemacie blokowym kodera **pętli sprzężenia zwrotnego**, w której następuje dekodowanie (dekompresja) próbek bloków. Pętla ta realizuje więc ścieżkę dekodowania danych. Hybrydowy koder obrazu zawiera więc w sobie znaczną część bloków funkcjonalnych dekodera, co wpływa w pewnym stopniu na jego złożoność.

4.30. Poziomy wyboru wariantów kompresji, czyli typy obrazów i typy bloków

Na barkach kodera obrazu spoczywa bardzo trudne zadanie wyboru najlepszego sposobu zakodowania treści obrazów. Chcemy, żeby wybór ten był uzależniony nie tylko od treści całych obrazów, ale dodatkowo, żeby również uwzględniał on to, jaki jest charakter tej treści w poszczególnych fragmentach danego obrazu. Z wymienionych oczekiwań wynikają więc dwa poziomy wyboru trybów kompresji treści: poziom bloków obrazu i poziom obrazu.

4.30.1. Poziom bloku wyboru wariantu kompresji

Pierwszym poziomem jest poziom bloków obrazu. W zależności od treści aktualnie kodowanego bloku, treści innych bloków obrazu, które z tym blokiem sąsiadują, czy wreszcie treści obrazów z innych chwil czasowych (w stosunku do obrazu aktualnie kodowanego) lepszy dla aktualnie kodowanego bloku może się okazać tryb kompresji wewnątrzobrazowej (INTRA), bądź międzyobrazowej (INTER). Lepszy, to znaczy pozwalający osiągnąć wyższą efektywność kompresji danych bloku. W oparciu o to kryterium koder zadecyduje więc, który z trybów kompresji najlepiej jest dla bloku wybrać: tryb INTRA, czy tryb INTER. W przypadku wyboru wariantu INTER należy jeszcze zadecydować, czy ma to być kodowanie międzyobrazowe z jednokierunkową predykcją bloku (czyli realizowane w oparciu o jeden obraz odniesienia, który w czasie znajduje się przed aktualnie kodowanym obrazem), czy może międzyobrazowe, ale z predykcją dwukierunkową (czyli przeprowadzane na bazie dwóch obrazów odniesienia – jednego z wcześniejszej i drugiego z późniejszej chwili czasowej, w stosunku do momentu wystąpienia

kodowanego obrazu). Z tak podejmowanych przez koder decyzji wynikają więc trzy główne typy bloków (przy czym jeszcze nie mówimy jaki jest rozmiar tych bloków):

- typ INTRA, nazywany typem **I** (I od ang. **I**ntra coding);
- typ **P**, oznaczający kodowanie predykcyjne międzyobrazowe z predykcją jednokierunkową (od ang. **P**redictive coding);
- typ **B**, mówiący o zastosowaniu kodowania predykcyjnego międzyobrazowego z dwukierunkową predykcją (od ang. **B**i-predictive coding).

Użycie w poszczególnych blokach obrazu określonego wariantu kompresji (warianty **I**, **P**, **B**) wymaga właściwego poinformowania dekodera obrazu o tym, jaki tryb kompresji został w bloku zastosowany. Z jednej strony chcielibyśmy móc o tym decydować już na poziomie bardzo małych bloków obrazu, żeby jak najlepiej dopasować się do charakteru kodowanej treści. Jednak z drugiej strony widzimy, że z tym wyborem wiąże się pewien koszt sygnalizacji trybu (bo do dekodera trzeba wysłać informację o użytym trybie). Pozostaje jeszcze kwestia złożoności obliczeniowej zastosowanego rozwiązania. Im większa paleta rozmiarów bloków, w których można podejmować decyzje o trybie kompresji, tym złożoność kodowania jest większa. Trzeba więc tutaj znaleźć pewien kompromis.

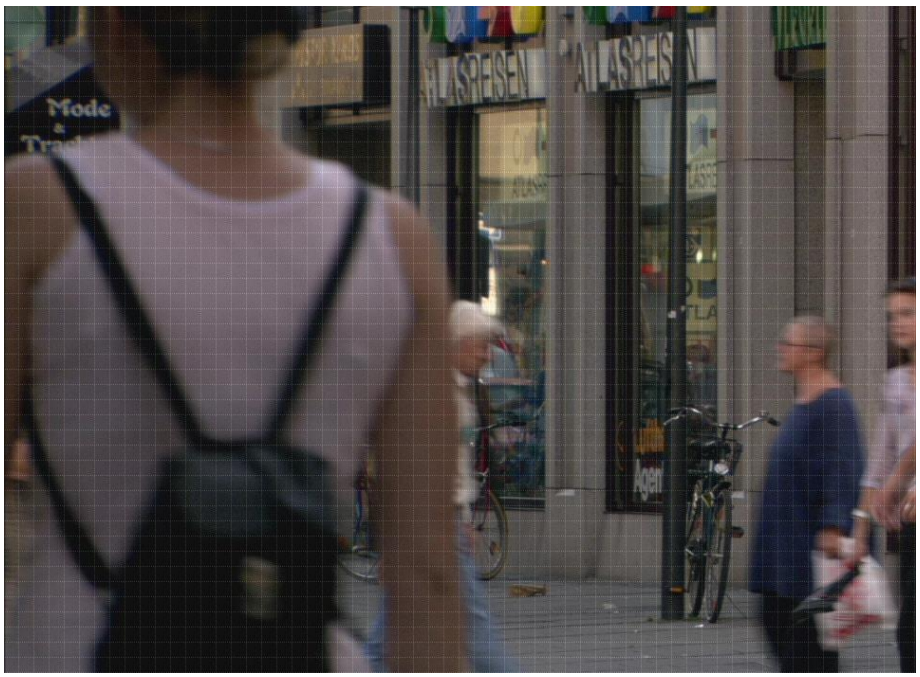
W starszych koderach obrazu (tych opracowanych jeszcze przed rokiem 2010) powszechnie stosowanym rozwiązaniem był wybór wariantów kompresji bloków (warianty **I**, **P**, **B**) w blokach o stałej wielkości 16x16 próbek obrazu (w praktyce 16x16 próbek składowej luminancji, plus skojarzone z nimi próbki dwóch składowych chrominancji). Bloki o tej wielkości nazywano **makroblokami**. Podział przykładowego obrazu na makrobloki został przedstawiony na rysunku 4-69. Takie rozwiązanie upraszczało oczywiście działanie kodera, jednak wraz z nim mniejsza była zdolność adaptacji sposobu kompresji treści w zależności od jej charakteru.

Rozwiązania dużo bardziej zaawansowane znajdują się w koderach najnowszych, opracowanych po roku 2010. Dobrym przykładem jest tutaj technologia HEVC kompresji ruchomego obrazu. W tej technologii nie ma już **makrobloków** o stałym rozmiarze. Zostały one zastąpione tzw. **jednostkami kodowania** (ang. coding units – **CU**), czyli blokami, które mogą przyjmować rozmiary w zakresie od 8x8, poprzez 16x16 i 32x32, aż po 64x64. To koder decyduje o tym, jakie mają być wielkości poszczególnych bloków CU, oraz dodatkowo, jaki wariant kompresji należy w tych blokach zastosować (INTRA czy INTER). Podział przykładowego obrazu na jednostki kodowania CU przedstawia rysunek 4-69.

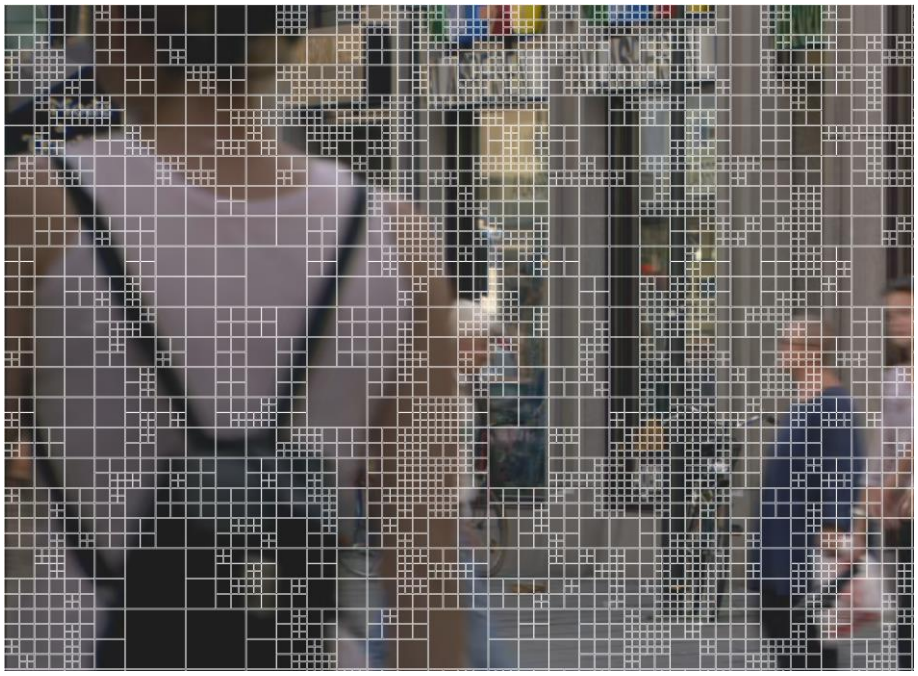
Na tym jednak praca kodera jeszcze się nie kończy. Na razie wybrano same warianty kompresji bloków (wariant **I**, **P** lub **B**). Bardziej szczegółowe pytanie brzmi: jak dalej przeprowadzić kompresję treści bloków w wybranych wariantach? W przypadku bloków typu **I** należy zadecydować, czy warto jest dokonać dalszego podziału bloku na jeszcze mniejsze fragmenty i dla każdego z takich fragmentów wybrać najlepszy predyktor treści (z puli dostępnych predyktorów). Podobnie jest w przypadku dwóch pozostałych typów bloków, czyli **P**, oraz **B**. Możliwy jest dalszy ich podział na mniejsze fragmenty (koder musi „pomyśleć”, jak to należy zrobić), tak żeby dla każdego z takich fragmentów niezależnie wyznaczyć wektor (lub wektory) ruchu i zrealizować kodowanie predykcyjne międzyobrazowe.

Na tym samym poziomie, co warianty kompresji bloków decyduje się również o sile kwantowania danych resztkowych predykcji treści (dane te są przedstawione w dziedzinie transformaty). Co makroblok lub odpowiednio, co jednostkę kodowania CU koder może więc zmienić poprzednio podjętą decyzję i dostosować stopień stratności kompresji (właśnie poprzez

wybór sposobu kwantowania danych) do charakteru danych, które stanowią treść bloku. Informacja o sposobie kwantowania sygnału jest oczywiście transmitowana do dekodera obrazu. Bez niej dekoder nie byłby w stanie poprawnie przeprowadzić operacji dekwantowania sygnału. Poziom makrobloku, czy jednostki CU, spełnia więc podwójną funkcję: to tutaj wybiera się wariant kompresji treści (INTRA bądź INTER) i ustawia parametry kwantowania danych.



podział obrazu na **makrobloki** – stały rozmiar bloków



podział obrazu na **jednostki kodowania CU** – zmienny rozmiar bloków

Rysunek 4-69. Podział obrazu na bloki, w których koder podejmuje decyzję o stosowanym wariancie kompresji ich treści: INTRA lub INTER.

4.30.2. Poziom obrazu wyboru wariantów kompresji

To jakie warianty kompresji bloków są możliwe do użycia w danym obrazie (uwaga: są możliwe, co nie znaczy, że na pewno w obrazie te warianty zostaną faktycznie wykorzystane) przekłada się na typ obrazu. W świetle zastosowania trzech wariantów kompresji bloków, wyróżnia się trzy typy obrazów. Są to obrazy **typu I**, **typu P** i **typu B**.

Obrazy pierwszego typu (**typ I**) zawierają tylko i wyłącznie makrobloki (lub jednostki CU) typu I. Obrazy tego typu są więc w całości kodowane z użyciem metod kompresji wewnątrzobrazowej.

W obrazach **typu P**, oprócz makrobloków (jednostek CU) typu I dopuszcza się dodatkowo możliwość użycia makrobloków (jednostek CU) typu P. W tych obrazach, oprócz kompresji wewnątrzobrazowej, można więc również zastosować algorytmy kompresji międzyobrazowej, ale tylko z użyciem predykcji jednokierunkowej.

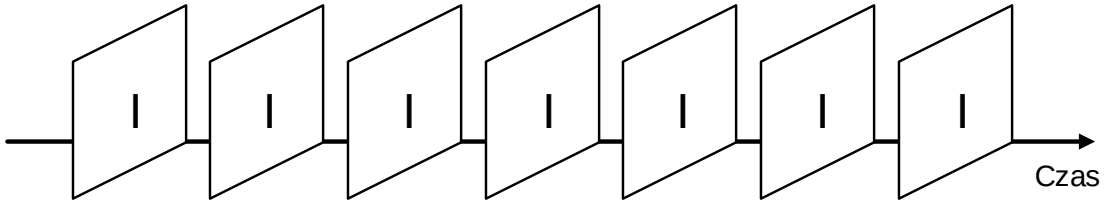
W obrazach **typu B** dopuszcza się wykorzystanie wszystkich omówionych wariantów kompresji bloków (makrobloków). Część bloków może zostać zakodowana w trybie I. Ale inne bloki mogą być już typu P lub B. Zastosowane typy bloków są oczywiście wynikiem decyzji kodera obrazu.

Należy raz jeszcze mocno tutaj podkreślić, że nie wszystkie dopuszczalne typy bloków muszą faktycznie wystąpić w obrazach P i B. To sam fakt możliwości użycia określonych wariantów kompresji bloków decyduje tutaj o typie obrazu.

4.31. Jak kodowana sekwencja dzielona jest na obrazy określonego typu?

Kodując obrazy sekwencji należy ustalić, jaki typ mają mieć kolejne jej obrazy. Typ I, P, czy może B? Przyjęte typy kolejnych obrazów zależą od zaplanowanego scenariusza kompresji sekwencji (np. silna kompresja, czy słabsza kompresja), który jest podyktowany wymaganiami, jakie stawia się przed system kompresji ruchomego obrazu w danym zastosowaniu.

Można powiedzieć, że rozwiązaniem najprostszym pod każdym względem jest przyjęcie typu I dla każdego z obrazów sekwencji (patrz rysunek 4-70). W każdym z obrazów stosowane są więc tylko narzędzia kompresji wewnątrzobrazowej. Te algorytmy są proste, przez to znacznie mniej wydajne od metod kodowania międzyobrazowego. Jednak niewątpliwym ich plusem jest niewielka złożoność kodowania i dekodowania danych. Korzyścią jest również i to, że w każdym momencie czasu możemy rozpocząć dekodowanie sekwencji. W końcu dekompresja danego obrazu nie wymaga znajomości innych obrazów.

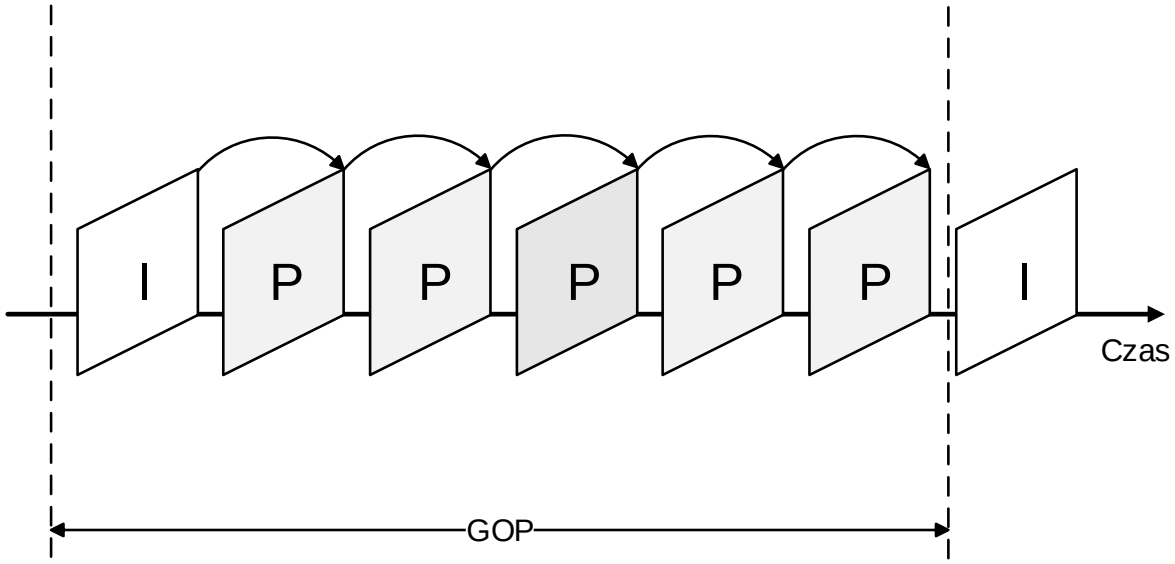


Rysunek 4-70. Przyjęcie typu I dla każdego z obrazów sekwencji.

Znacznie wydajniejszym rozwiązaniem od wyżej przedstawionego jest „dopuszczenie do głosu” również obrazów typu P. Rozpoczynając kodowanie sekwencji, pierwszy obraz musimy oczywiście oznaczyć jako typu I. Ale kolejne obrazy mogłyby już być typu P. Dysponując dodatkowo narzędziami kompresji międzyobrazowej jednokierunkowej obrazy typu P efektywniej zakodują daną treść niż zdołałby to zrobić obraz I. Z tej perspektywy, poza pierwszym obrazem, wszystkie pozostałe chcielibyśmy już kodować jako typu P.

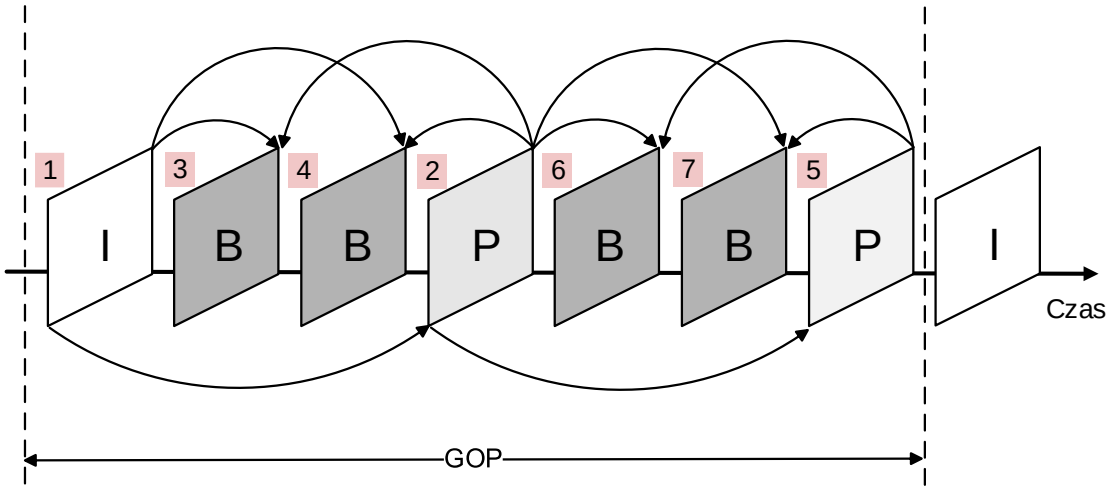
Jednak w tym miejscu należy sobie uzmysłowić, że w praktyce aż tak daleko posunąć się nie możemy. Oznaczenie tylko pierwszego obrazu jako typu I byłoby rozwiązaniem dość mocno niepraktycznym. Poprawne zdekodowanie dowolnego obrazu P wymaga znajomości treści innego obrazu, który w czasie występuje przed obrazem, który aktualnie dekodujemy. A więc, gdybyśmy rozpoczęli dekodowanie już po czasie, w którym wystąpił pierwszy obraz (który jest typu I) to żadnego z obrazów P nie bylibyśmy już w stanie poprawnie zdekodować. Brakowałoby nam któregoś z obrazów (I lub P), który jest podstawą poprawnej dekompresji obrazu, który aktualnie dekodujemy.

Dlatego, z powyższego powodu, jesteśmy żywo zainteresowani tym, żeby co jakiś czas umieszczać w zakodowanej sekwencji obraz typu I. Chcemy tak robić pomimo tego, że obrazy I pochłaniają znacznie więcej bitów niż obrazy typu P. Jednak korzyścią z takiego rozwiązania jest to, że obrazy I stanowią punkt dostępowy do zakodowanej sekwencji – od tych obrazów możemy rozpocząć dekodowanie zakodowanych obrazów. Podczas kompresji sekwencja jest więc dzielona na pewne segmenty (patrz rysunek 4-71). Każdy z segmentów, nazywany **grupą obrazów** (ang. group of pictures – GOP) rozpoczyna się obrazem typu I, po którym następują obrazy innych typów, np. typu P. Kolejny obraz typu I rozpoczyna już więc nową grupę obrazów. Zastosowana długość GOP (mierzona liczbą obrazów, które wchodzą w jej skład lub adekwatnie odległością dwóch kolejnych obrazów typu I w zakodowanej sekwencji) determinuje efektywność kompresji danych, ale również to, w których momentach czasu możemy rozpocząć dekodowanie obrazów. Jej wartość jest więc wynikiem kompromisu pomiędzy wskazanymi parametrami i uwzględnia docelowy obszar zastosowań kompresji. Na przykład, w przypadku transmisji obrazów na potrzeby telewizji cyfrowej typowa długość GOP odpowiada czasie 0,5 sekundy sekwencji – zatem dekoder użytkownika telewizji cyfrowej rozpocznie dekodowanie obrazów najpóźniej w ciągu 0,5 sekundy od jego załączenia (uwaga: pomija się tutaj oczywiście czasy potrzebne na transmisję obrazów do dekodera, czy uruchomienie samego urządzenia).



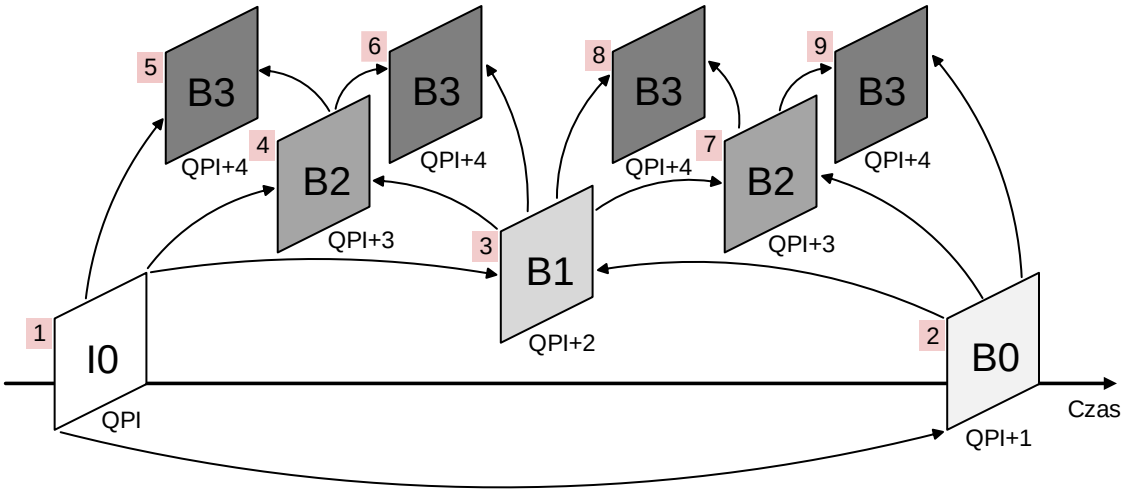
Rysunek 4-71. Przyjęcie struktury IPPPP… dla grupy obrazów.

Jeszcze wyższą efektywność kompresji obrazów gwarantuje zastosowanie, obok obrazów typu I i P, również obrazów typu B. W każdej grupie obrazów występują więc obrazy wszystkich trzech typów. Możliwość zastosowania w obrazach B narzędzi kodowania z predykcją międzyobrazową dwukierunkową istotnie zwiększa efektywność kompresji danych. Ale rodzi też szereg niewystępujących w obrazach I i P problemów, jak chociażby opóźnienie kodowania i dekodowania obrazów, czy wyższa złożoność kompresji i dekompresji danych (więcej na ten temat w osobnych punktach 4.33 i 4.34). Z punktu widzenia zastosowania praktycznego dość często przyjmowanym kompromisem jest skorzystanie ze struktury GOP jak na rysunku 4-72. Taki schemat kompresji (lub schemat do niego podobny) znajduje zastosowanie na przykład w systemach telewizji cyfrowej.



Rysunek 4-72. Przyjęcie struktury IBBPBBP… dla grupy obrazów. Numery przy obrazach wyznaczają kolejność kodowania obrazów.

W ostatnich latach (po roku 2010) bardzo popularna stała się jeszcze inna praktyka wykorzystania obrazów typu B. Mowa o strukturze kodowania obrazów z **hierarchicznymi obrazami B** [HEVC, HEVCbook]. W przypadku tej struktury stosuje się jeszcze inną niż wcześniej kolejność kodowania obrazów, przy czym jest ona taka jak na zamieszczonym rysunku 4-73 (patrz numery w czerwonym okienku). W omawianej strukturze obrazy przypisuje się do jednego z czterech poziomów hierarchii numerowanych od 0 do 3 (stąd właśnie cyfry na rysunku 4-73 w oznaczeniu typu obrazu). Co warte zauważenia, istotną cechą charakterystyczną tej struktury są stosowane wartości parametru kwantyzacji QP dla poszczególnych obrazów – w stosunku do parametru QP obrazu typu I (oznaczonego na rysunku jako I0) obserwujemy tutaj wzrost wartości parametru o 1, jeśli zmienimy typ obrazu na tym samym poziomie hierarchii, bądź też numer poziomu hierarchii. Strukturę kodowania obrazów z hierarchicznymi obrazami B cechuje bardzo wysoka efektywność kompresji obrazów, dlatego znajduje ona zastosowanie w koderach wizyjnych najnowszych generacji, np. HEVC.



Rysunek 4-73. Kodowanie obrazów sekwencji z użyciem hierarchicznych obrazów B. Numery znajdujące się po lewej stronie obrazów wyznaczają kolejność ich kodowania.

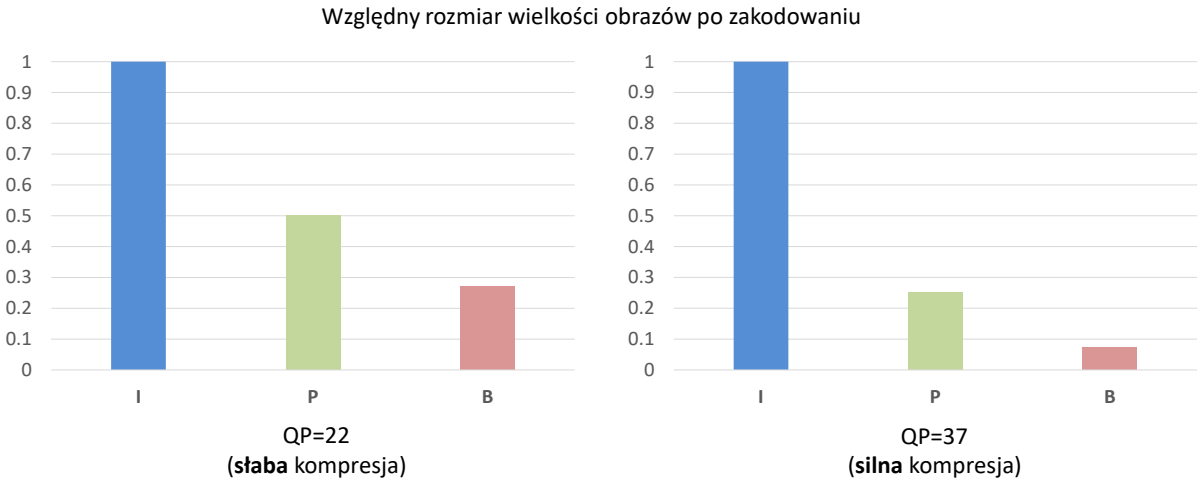
Uwaga: Dziwić może fakt, że ostatni z przedstawionych obrazów nie jest obrazem P, tylko B. Nowa technologia HEVC kodowania obrazu wprowadza pojęcie **uogólnionego obrazu typu P i B**. Chociaż taki obraz korzysta z tylko jednego obrazu odniesienia (na rysunku 4-73 jest to obraz oznaczony jako I0), to podstawą predykcji kodowanego bloku mogą być dwa bloki odniesienia, a nie tylko jeden. Blok obrazu B0 może być więc kodowany z użyciem dwóch różnych wektorów ruchu, które odnoszą się jednak do tego samego obrazu odniesienia. W tym przypadku nie jest to więc typowa predykcja dwukierunkowa, która odnosi się zarówno do przeszłych jak i przyszłych chwil czasowych.

4.32. Efektywność kompresji obrazów poszczególnych typów

Najwyższa efektywność kompresji danych jest domeną obrazów typu B. Źródłem tej efektywności, poza samą kompresją stratną, jest zastosowanie dwukierunkowej, międzyobrazowej predykcji próbek bloków. Jak wiemy ten typ predykcji treści cechuje się bardzo wysoką skutecznością, przez co wynikowy błąd przewidywania kodowanych próbek przyjmuje zwykle

bardzo małe wartości. Tak małe, że sygnał ten udaje się później opisać już przy niskich nakładach bitowych. Dlatego po kompresji rozmiar obrazu typu B jest zdecydowanie mniejszy, w porównaniu z rozmiarami obrazów I oraz P. Spośród trzech wymienionych typów obrazów najniższą efektywność kompresji danych notuje się dla obrazów I.

Przedstawienie dokładnej relacji rozmiarów trzech typów obrazów (I, P, B) wymaga uwzględnienia szeregu istotnych czynników, które na ten wynik mają wpływ. Treść kodowanych obrazów, zastosowana technologia kompresji danych wraz z użytymi narzędziami kodowania (koder starszy czy nowszy, plus użyte tryby kompresji oraz sposób ich wyboru), scenariusz kodowania obrazów (silna kompresja czy niski stopień kompresji), itd., to są te czynniki. Doświadczenie autora pokazuje jednak, że w przypadku współczesnych koderów wizyjnych (np. kodera AVC), skonfigurowanych w sposób pozwalający na osiągnięcie wysokiej efektywności kompresji danych, można podać następującą, dość ogólną regułę: obrazy typu P są **2-4 krotnie** mniejsze niż obrazy I, podczas gdy obrazy B są **2-3,5 razy** mniejsze od obrazów P. Co ciekawe, wraz ze wzrostem stopnia kompresji danych (wzrost parametru QP) różnice wielkości obrazów się pogłębiają. Dokładne relacje wielkości trzech typów obrazów, jakie dla typowej sekwencji testowej zostały uzyskane w przypadku oprogramowania **JM 18.5** kodera AVC, przedstawiono na poniższych dwóch wykresach.



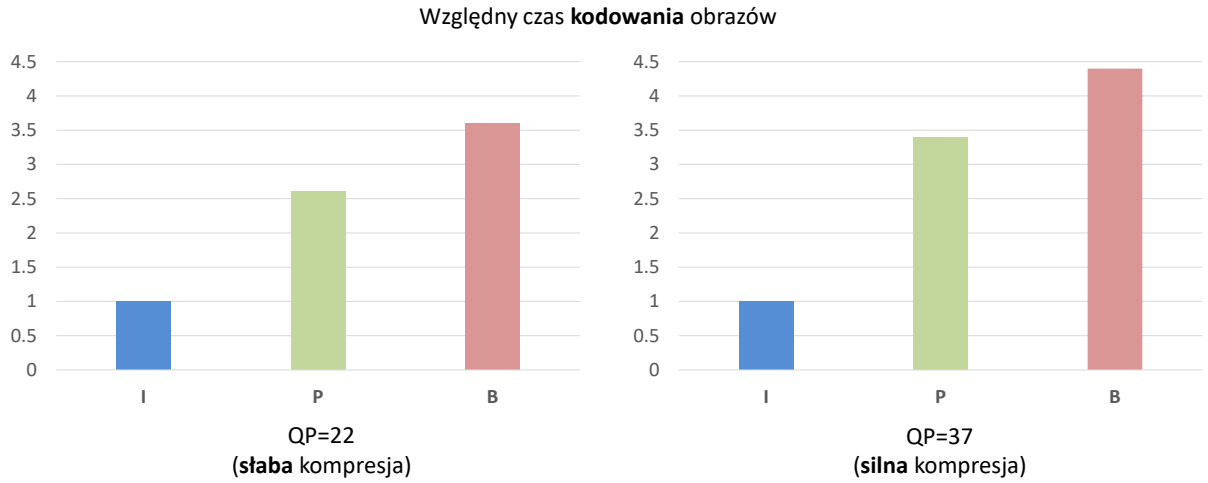
Rysunek 4-74. Względny rozmiar obrazów danego typu po kompresji. Wyniki dla oprogramowania JM 18.5 kodera AVC, dla dwóch różnych ustawień kodera, odpowiadających kompresji słabej oraz silnej.

Wysoka efektywność kodowania obrazów B przekłada się na wzrost efektywności kompresji całej sekwencji. Na dokładne rezultaty kompresji wpływ ma liczba użytych obrazów B w GOP oraz sposób rozmieszczenia tych obrazów wewnątrz grupy.

4.33. Złożoność kodowania i dekodowania obrazów

Mające zastosowanie w obrazach poszczególnych typów narzędzia kompresji decydują o złożoności kodowania i dekodowania obrazów. Jednak, co ciekawe, ogólne wnioski złożoności metod mogą być inne w przypadku kodera i dekodera obrazu.

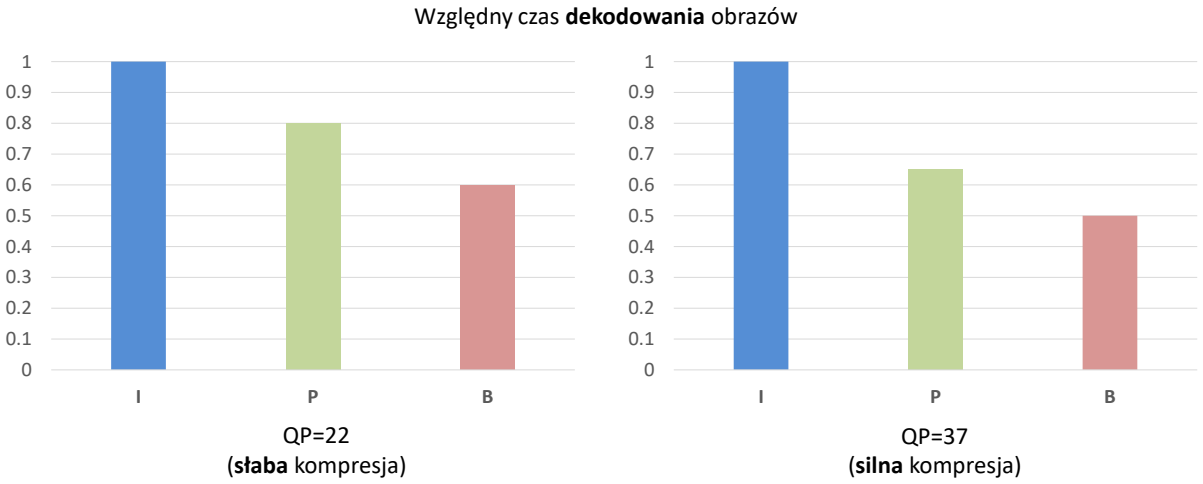
Rozpocznijmy więc analizę od strony kodującej. Tutaj, zdecydowanie najprostsze są metody kompresji wewnątrzobrazowej. Predykcja treści kodowanego bloku, dokonywana w oparciu o wybrane próbki sąsiednich bloków, które pochodzą z aktualnie kodowanego obrazu nie nastręcza problemów obliczeniowych. Czasy kodowania obrazów typu I są więc najkrótsze. W obrazach typu P, oprócz metod kompresji wewnątrzobrazowej, dochodzą jeszcze te, które wymagają przeprowadzenia bardzo trudnej obliczeniowo estymacji ruchu w sekwencji. Mowa o algorytmach kodowania z międzyobrazową predykcją treści. Zastosowanie tych ostatnich algorytmów sprawia, że złożoność kodowania obrazów P jest zauważalnie wyższa niż obrazów I. Jednak pod tym względem jeszcze trudniejsze są obrazy B. W tych obrazach może zachodzić konieczność aż dwukrotnego wykonania „ciężkiej” obliczeniowo estymacja ruchu w ramach kodowanego bloku obrazu (dwukrotnego, bo na potrzeby predykcji międzyobrazowej dwukierunkowej), co czyni te obrazy najtrudniejszymi do zakodowania. Typowe relacje nakładów obliczeniowych potrzebnych na zakodowanie obrazów I, P, i B są takie jak na przedstawionych poniżej rysunkach. Są to wyniki, jakie autor uzyskał w przypadku oprogramowania **JM 18.5** kodera **AVC**, które zostało uruchomione na procesorze komputera. Dane te należy więc traktować jako orientacyjne, pamiętając, że na szczegółowe wyniki wpływ ma szereg czynników, jak technologia kompresji (koder starszy, np. MPEG-2, koder nowszy, np. AVC), czy sposób przygotowania oprogramowania kodera (stopień optymalizacji kodu programu).



Rysunek 4-75. Względny czas **kodowania** obrazów danego typu. Wyniki dla oprogramowania JM 18.5 kodera AVC, dla dwóch różnych ustawień kodera, odpowiadających kompresji słabej oraz silnej.

Tej bardzo trudnej obliczeniowo estymacji ruchu nie ma jednak po stronie dekodera obrazu. Tutaj dekoder otrzymuje od kodera już gotową informację o tym, w jaki sposób przewidywać próbki bloku oraz skąd, w obrazie odniesienia lub obrazie aktualnie kodowanym, pobierać dane do predykcji treści bloku, który został wcześniej zakodowany w trybach I, P lub B. Może się więc okazać, że sposób predykcji treści bloku, INTRA bądź INTER, nie będzie mieć po stronie dekodera obrazu większego znaczenia, z punktu widzenia czasu wykonania tej operacji (operacji przewidywania treści). Czasochłonność czynności dekodera, które wchodzą w skład dekodowania transformatowego, powinna być również zbliżona, bez względu na to, czy zastosowano tryb INTRA czy INTER. Wyjątkiem jest tutaj niewątpliwie sama dekompresja entropijna danych, czyli pierwszy etap dekodowania obrazu. A wynika to wprost z faktu silnego zróżnicowania ilości danych, które w przypadku trybów (obrazów) I, P, i B dekodowaniu entropijnemu są poddawane. Dlatego zdaniem autora, to właśnie różnica w czasie działania bloku

dekodera entropijnego w dekoderze obrazów I, P i B, jest tym czynnikiem, który w największym stopniu wpływa na zróżnicowanie całościowego czasu dekodowania obrazów I, P i B. W obrazach B, z uwagi na uzyskiwaną bardzo wysoką efektywność kompresji obrazów, danych, które są poddawane entropijne dekompresji jest stosunkowo najmniej. Zatem dane te udaje się zdekodować w relatywnie najkrótszym czasie (w porównaniu z obrazami innych typów). W obrazach P, w porównaniu z obrazami B, tych danych jest już więcej, co wydłuża czas działania dekodera entropijnego. A jeszcze więcej pracy niż w przypadku obrazów P i B, ma dekoder entropijny, który działa w ramach obrazów I. Objętość danych, które opisują obrazy I jest, jak wiadomo, zdecydowanie największa. Z przedstawionych tutaj reguł, oraz wzajemnej relacji wielkości obrazów danego typu (I, P i B), wynikają związki pomiędzy czasami dekodowania poszczególnych typów obrazów. Typowe relacje tych czasów są takie, jak na przedstawionych wykresach, które prezentują wyniki, jakie autor książki otrzymał w przypadku modelowego oprogramowania **JM 18.5** dekodera **AVC**. Proszę zauważyć, iż co do samego charakteru otrzymanych wartości, są one w przypadku dekodera dokładnie odwrotne niż miało to miejsce podczas kodowania obrazów.



Rysunek 4-76. Względny czas **dekodowania** obrazów danego typu. Wyniki dla oprogramowania JM 18.5 dekodera AVC, dla dwóch różnych scenariuszy, odpowiadających kompresji słabej oraz silnej.

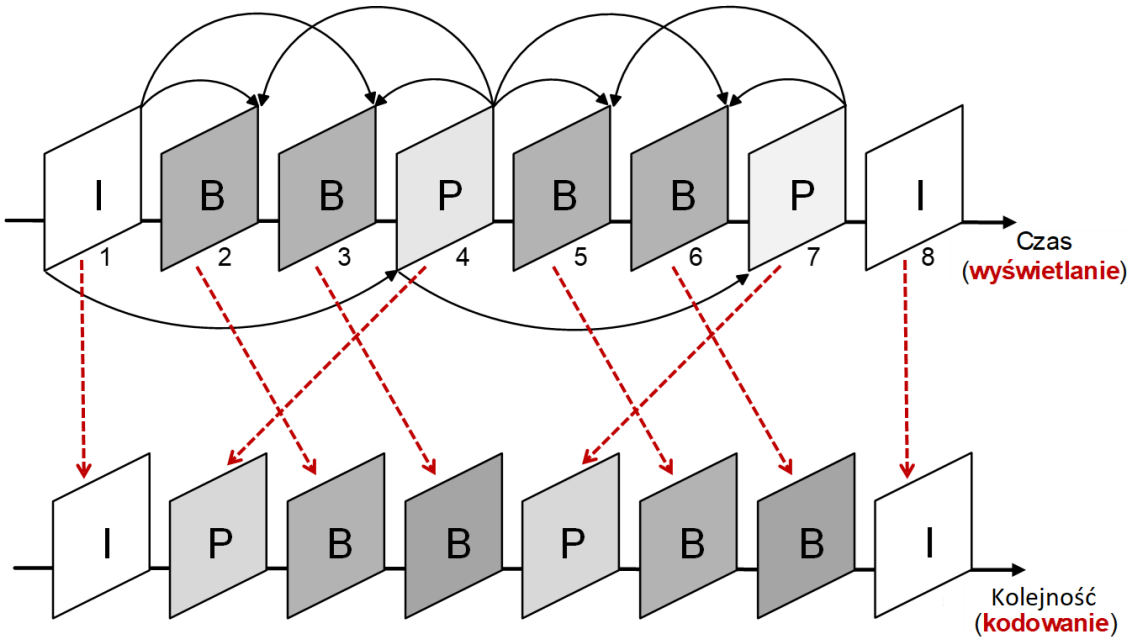
Oprócz samych czasów działania kodera czy dekodera obrazu istotna jest również tzw. złożoność pamięciowa algorytmów. Złożoność ta odnosi się do ilości pamięci, jaką program kodera, czy dekodera obrazu wykorzystuje podczas swojej pracy. W tej materii można podać następującą ogólną regułę: zapotrzebowanie na pamięć jest zdecydowanie największe w przypadku obrazów B, mniejsze przy obrazach P, a najmniejsze podczas kodowania czy dekodowania obrazów typu I. Podane związki wprost wynikają z mechanizmów predykcji treści bloków, które można stosować w obrazach I, P, czy B. W przypadku obrazów P czy B mechanizmy te mogą, podczas przewidywania, czynić odwołania również do obrazów z innych chwil czasowych. Zdekodowane wersje obrazów z tych innych chwil czasowych należy więc w koderze i dekoderze obrazu zapamiętać. Nie robi się tego oczywiście w przypadku obrazów I – tutaj zapamiętuje się tylko część danych, które w aktualnie kodowanym, bądź dekodowanym obrazie, zostały już przetworzone. Mechanizm predykcji dwukierunkowej międzyobrazowej, obecny w obrazach B, stwarza potrzebę zapamiętywania większej liczby obrazów odniesienia niż ma to miejsce w przypadku obrazów P.

4.34. Kolejność kodowania a kolejność wyświetlania obrazów sekwencji

Narzędzia kompresji danych, które są stosowane w obrazach I oraz P nie wymagają zmiany kolejności kodowania obrazów, w stosunku do porządku, zgodnie z którym są one wyświetlane później na ekranie widza. Jednak taka zmiana jest konieczna w przypadku obrazów typu B. Wynika to wprost z istoty mechanizmu dwukierunkowej międzyobrazowej predykcji treści.

Przewidując z użyciem tej predykcji treść obrazu z chwili czasowej 2 (obraz oznaczony numerem 2 – patrz górna część rysunku 4-77) należy się odwołać do treści zdekodowanych i zapamiętanych wersji dwóch obrazów: z chwil czasu 1 oraz 4. Zanim przystąpimy więc do kodowania pierwszego obrazu B (zgodnie z rysunkiem), to wcześniej trzeba zakodować, odtworzyć (zdekodować) i zapamiętać wynik dekodowania dla obrazów z chwil 1 i 4. To samo dotyczy kodowania obrazu B z chwili 3. Analogicznie, kodowanie obrazu z chwili 5 musi być poprzedzone wcześniejszą kompresją, dekompresją i zapamiętaniem wyniku dekompresji dla obrazów z chwil 4 oraz 7. W przypadku użycia obrazów B kolejność kodowania obrazów nie jest więc tożsama z kolejnością w jakiej są one później prezentowane widzowi. Dodatkowo, określone obrazy (ich wersje zdekodowane), które będą odniesieniem dla predykcji innego obrazu, muszą być zapamiętywane, czyli konieczne jest buforowanie wskazanych obrazów. Ponieważ kodowanie danego obrazu B należy wstrzymać dopóki nie zostaną zakodowane jego obrazy odniesienia (z czego jeden z tych obrazów pochodzi z przyszłej chwili czasowej, w stosunku do chwili wystąpienia kodowanego obrazu), to mówi się, że obrazy B wprowadzają **opóźnienie kodowania**.

Problem opóźnienia kodowania dobrze widać na poniższym rysunku. Inna niż kolejność wyświetlania, kolejność kodowania obrazów, komplikuje więc pracę kodera obrazów.



Rysunek 4-77. Kolejność kodowania, a kolejność wyświetlania obrazów sekwencji. Stosowanie obrazów B wymusza ściśle określoną kolejność kodowania obrazów. **Uwaga:** Liczby pod obrazami zostały określone zgodnie z kolejnością, w jakiej obrazy te są wyświetlane na ekranie widza.

Wymienione powyżej aspekty dotyczą również dekodera obrazów.

4.35. Złożoność hybrydowego kodera i dekodera obrazów

W przypadku technologii hybrydowej kodowania danych istnieje silna asymetria złożoności kodera i dekodera obrazów. Tutaj, koder jest kilka-kilkadziesiąt razy bardziej obliczeniowo złożony niż dekoder (w przypadku programowej, szybkiej realizacji nowego kodeka HEVC czas działania kodera jest aż około **20 razy dłuższy** od czasu dekodowania sekwencji!!!). Wynika to z szeregu czynników, wśród których z pewnością należy wymienić:

1. Estymacja ruchu obiektów w sekwencji

Wykonywana na potrzeby międzyobrazowej predykcji treści jest bardzo złożona obliczeniowo. Występuje tylko po stronie kodera (nie ma jej w dekoderze). Pomimo zastosowania szybkich metod stanowi ona w koderze od 30% do 70% wszystkich jego obliczeń. W praktyce jest to więc najtrudniejszy (lub jeden z najtrudniejszych) etap obliczeniowy kodera. Wynikiem estymacji ruchu są wektory ruchu, które wskazują na miejsce położenia w obrazie odniesienia bloków, które w największym stopniu przypominają te, które kodujemy. Dekoder otrzymuje (od kodera) wektory ruchu jako gotową informację – ich zdekodowanie nie nastręcza już obliczeniowych problemów w dekoderze.

2. Mechanizm wyboru trybów kodowania bloków obrazu

Współczesny koder wizyjny dysponuje bardzo obszernym wachlarzem trybów kodowania treści. Dzięki temu obrazy możemy reprezentować bardzo wydajnie, ale dodatkową konsekwencją jest długi czas kodowania treści. W jak dużych blokach obrazu decydować o trybie kompresji (INTRA lub INTER) treści tych bloków, w blokach o jakim rozmiarze najlepiej jest przewidywać treść obrazu, który z dostępnych wariantów predykcji treści użyć w danym bloku, jakie powinny być składowe wektorów ruchu, z jaką dokładnością należy reprezentować poszczególne składowe częstotliwościowe sygnału błędu przewidywania treści (czyli jak kwantować próbki widma sygnału błędu predykcji treści), itd. To są pytania, na które musi odpowiedzieć sobie koder. W praktyce więc, określony fragment obrazu daje się zakodować na olbrzymią liczbę różnych sposobów. Podjęcie decyzji o sposobie kodowania określonego fragmentu treści wymaga z reguły jego wielokrotnego zakodowania, za każdym razem z użyciem innego wariantu kompresji. Nakład obliczeniowy tej czynności jest zatem bardzo duży. Czas działania kodera obrazu, ale również jego efektywność, będą silnie zależeć od tego, w jaki sposób koder podejmuje swoje decyzje.

Proszę zauważyć, że tego „trudnego” obliczeniowo etapu wyboru trybów w ogóle nie ma po stronie dekodera obrazu. Dekoder otrzymuje od kodera już gotową informację o tym, jakie tryby kompresji zostały zastosowane w ramach kolejnych fragmentów obrazu. Realizuje więc pojedynczy etap obliczeniowy dekompresji danych.

3. Rekonstrukcja (dekodowania) zakodowanych bloków

Żeby przewidywanie treści obrazu przebiegało dokładnie tak samo po stronie kodera i dekodera, to wykonujący tę operację koder musi bazować na wersji danych znanych później dekoderowi. Czyli na zdekodowanej, a nie oryginalnej wersji treści. Z tego powodu zachodzi więc w koderze konieczność dekompresji dopiero co zakodowanych danych. Koder, oprócz etapów kodowania, realizuje zatem również część bloków funkcjonalnych dekodera, co zwiększa niewątpliwie jego złożoność.

W opisywanej sytuacji cieszy jednak fakt, że ten ogromny „ciężar” obliczeń spada na koder, a nie na dekoder obrazu. Sekwencję obrazów możemy zakodować raz, używając do tego celu wysokowydajnej obliczeniowo farmy komputerów. Natomiast dekodowanie jest realizowane wiele razy, w urządzeniach użytkowników telewizji i Internetu. Dzięki relatywnie niskiej złożoności dekodowania urządzenia te nie muszą mieć wyśrubowanych parametrów technicznych, co przekłada się na ich niewygórowaną cenę.

Część III – Sterowanie koderem obrazu

4.36. Wprowadzenie

Koder obrazu, szczególnie ten współczesny, dysponuje obszernym wachlarzem wariantów (trybów) kompresji bloków. Do tego dochodzi jeszcze kwestia doboru sposobu (siły) kwantowania resztkowych danych predykcji kodowanych bloków. Powyższe wybory kodera różnią się między sobą efektywnością kompresji danych obrazu, na którą wpływ ma dodatkowo charakter treści bloków, które podlegają kodowaniu. Dlatego w tym miejscu pojawia się trudne pytanie o sposób zarządzania pracą kodera, czyli metodę wyboru trybów kompresji i siły kwantowania danych w poszczególnych blokach obrazu. Ile bitów przeznaczyć na zakodowanie poszczególnych bloków obrazu, jak te bloki najlepiej jest zakodować, żeby w przypadku stratnej kompresji obrazu osiągnąć pożądaną relację pomiędzy jakością obrazu a kosztem bitowym jego reprezentacji? To są dylematy, przed którymi stoi koder.

Z powyższej perspektywy, wchodzące w skład kodera narzędzia kompresji, są niczym innym jak zawierający dużą liczbę instrumentów samolot bez pilota. Z punktu widzenia praktycznego wykorzystania metod kompresji, w koderze potrzebny jest jeszcze jeden dodatkowy mechanizm, który podpowie koderowi sposób kodowania treści bloków. Mechanizm ten jest nazywany **blokiem sterowania koderem**. Od zastosowanych w tym bloku rozwiązań zależy w ogromnej mierze wydajność całego kodera obrazu.

W związku z powyższym faktem mechanizm sterowania koderem jest niezwykle ważnym składnikiem każdego kodera obrazu. Z tego powodu wydajne sposoby „pilotowania” pracą kodera są od wielu już lat ważnym tematem zainteresowania projektantów koderów. W ich toku opracowano szereg szczegółowych rozwiązań, dedykowanych dla kolejnych technik kompresji obrazów (MPEG-2, AVC, HEVC, itd..). Szczegóły tych rozwiązań można znaleźć w literaturze naukowej. W tej książce, ze względu na założone jej ramy oraz przeznaczenie, autor skupi się jedynie na fundamentalnych aspektach z tematyki sterowania koderami, które są prawdziwe zarówno w przypadku koderów starszych jak i tych najnowszych.

4.37. Jakie są najważniejsze cele sterowania koderem obrazu?

Wybierając tryby kodowania bloków chcemy doprowadzić do takiej sytuacji, w której wynikowy strumień zakodowanych danych będzie spełniał pożądane przez nas właściwości. Sterowanie koderem ma to zapewnić. Z jednej strony, pomimo zastosowania metod kompresji stratnej chcielibyśmy, żeby zakodowane obrazy miały jak najwyższą jakość. Z drugiej strony oczekujemy, że w trakcie kodowania uda nam się silnie zredukować objętość danych, które opisują kolejne obrazy. Jednoczesne osiągnięcie tych dwóch celów nie jest oczywiście możliwe. Tutaj, jak jeden parametr będzie rósł (np. stopień redukcji ilości danych w trakcie kodowania), to drugi będzie malał (jakość zakodowanych obrazów). W praktyce sterowania koderem obrazu wspomniane

właściwości wynikowego strumienia dotyczą więc albo prędkości bitowej zakodowanych danych, albo jakości materiału po kompresji, czy wreszcie próby zbalansowania jednego i drugiego parametru jednocześnie.

W pierwszym z wymienionych przypadków celem jest uzyskanie jak najlepszej jakości obrazów dla zadanej prędkości bitowej strumienia zakodowanych danych. Otrzymanie na wyjściu kodera zadanej wartości prędkości bitowej danych jest więc w tym przypadku nadrzędnym celem. Dlatego ten typ sterowania jest określany jako **sterowanie prędkością bitową**. W przypadku drugim celem jest otrzymanie najmniejszej prędkości bitowej przy założonej jakości obrazów po kompresji. Tutaj sterowanie skupia się więc na uzyskaniu zadanej jakości obrazów, po ich zakodowaniu. Mówimy więc o **sterowaniu jakością obrazów**. Ostatni, trzeci przypadek, to próba znalezienia najlepszego kompromisu pomiędzy prędkością bitową zakodowanych danych oraz jakością obrazów po kompresji. Celem jest więc jednoczesna optymalizacja prędkości bitowej i jakości zakodowanych obrazów. O tym typie sterowania mówi się, że jest to **optymalizacja prędkość bitowa – poziom zniekształceń** (lub alternatywnie, poziom jakości zakodowanych obrazów) w zakodowanych obrazach (ang. rate-distortion optimization).

4.38. W jaki sposób decydować o trybach kodowania bloków oraz sile kwantowania danych?

Bez względu jednak na to, który z typów sterowania koderem jest w danym zastosowaniu realizowany pozostaje pytanie, jak wybierać tryby kompresji bloków. Koder musi podjąć decyzję o tym, w których blokach obrazu zastosować kodowanie **INTRA**, a w których **INTER**, i jeśli ma taką możliwość, jakie mają być rozmiary tych bloków. Dodatkowo, koder decyduje w jak dużych blokach najlepiej jest przewidywać próbki kodowanego obrazu, który z dostępnych schematów predykcji użyć (w przypadku kodowania **INTRA** – który z predyktorów treści, w przypadku **INTER** – predykcja jedno- czy dwukierunkowa i jakie obrazy odniesienia), w jak dużych blokach zrealizować kodowanie transformatowe sygnału błędu predykcji treści i jak silnie kwantować dane, itd. Wszystko to generuje naprawdę ogromną liczbę kombinacji możliwych wyborów.

Najprostszym ideowo sposobem realizacji tego trudnego zadania byłoby zakodowanie obrazu, całego obrazu, na wszystkie możliwe sposoby, czyli z użyciem wszystkich dostępnych kombinacji kompresji treści. W każdym z przypadków należałoby ocenić efekty końcowe kodowania (stopień kompresji danych i jakość materiału po zakodowaniu) i na tej podstawie decydować, które z trybów faktycznie zastosować w poszczególnych blokach. Proszę jednak zobaczyć jak karkołomne obliczeniowo byłoby to zadanie. Tutaj, oprócz tego, że sama już liczba kombinacji zakodowania pojedynczego bloku obrazu jest bardzo duża, to z uwagi na predykcyjne kodowanie kolejnych bloków i obrazów, wynik kodowania danego bloku rzutuje na sposób kompresji bloków i obrazów kodowanych później od niego. W efekcie, w przypadku takiego podejścia, sumaryczna liczba możliwych kombinacji zakodowania jednego obrazu jest wprost niemożliwa do wyobrażenia. Z punktu więc widzenia trudu obliczeniowego tak wybierać trybów w praktyce się nie da.

Dlatego też optymalizację wyboru trybów należy przeprowadzać na poziomie bloków, np. makrobloków czy jednostek CU (tak jak w koderze HEVC), a nie na poziomie całego obrazu. W tym przypadku kompresja każdego z bloków na wszystkie możliwe sposoby zaczyna być już obliczeniowo możliwa (chociaż czas kodowania byłby tutaj ciągle bardzo długi), możliwa szczególnie w odniesieniu do starszych koderów. W tym przypadku nie sprawdzamy już, jak sposób zakodowania bloku przekłada się na możliwości kompresji kolejnych bloków, i następnych obrazów, co jest już bardzo znaczącym uproszczeniem. Ale, jak należy się spodziewać, prowadzi

również do pewnego spadku efektywności kompresji danych obrazowych. Jednak w najnowszych koderach obrazu, np. koder HEVC, liczba dostępnych wariantów zakodowania jednostki CU jest już tak duża, że i tego uproszczonego rozwiązania wyboru trybów nie da się użyć w praktyce. Również w przypadku tego podejścia potrzebne są dalsze „ustępstwa”.

Jednym z takich ustępstw może być rezygnacja z wykonania pełnego kodowania bloku danym trybem. Zamiast przechodzić pełną ścieżkę kompresji treści (predykcja próbek + kodowanie transformatowe + kodowanie entropijne) możemy np. całkowicie zrezygnować z kodowania entropijnego danych (**uwaga:** ale tylko na etapie testowania trybów!!!). Dodatkowo, badając efektywność kodowania transformatowego resztkowych danych predykcji, możemy zrezygnować z tradycyjnej transformacji typu DCT czy DST na rzecz prostszego obliczeniowo przekształcenia Walsha-Hadamarda. Na etapie oceny zasadności użycia danego trybu możemy z takich uproszczeń faktycznie skorzystać. Wydajność danego wariantu kompresji jest wtedy oceniana trochę bardziej zgrubnie, a nie bardzo, bardzo dokładnie.

Inną cenną wskazówką dla szybszego wyboru trybów w koderze jest informacja o statystykach wykorzystania poszczególnych trybów kodera, oraz wiedza o tym na ile użycie poszczególnych trybów przynosi zyski w postaci zwiększonej efektywności kompresji. Chodzi tutaj o całkowite pominięcie z procedury sprawdzania trybów tych wariantów kompresji, które i tak są przez koder rzadko wybierane, i w niewielkim stosunkowo stopniu przyczyniają się do wzrostu efektywności kodowania obrazu. Wiadomo na przykład, że na tej podstawie można zrezygnować z kodowania, które jest realizowane w najmniejszych blokach obrazu (np. 4x4). Jak pokazuje doświadczenie autora, sama tylko elimanacja tak wskazanych wariantów kodowania może przyspieszyć koder co najmniej dwukrotnie, nie powodując przy tym spadku efektywności kompresji o więcej niż 2% [Weg18]. Tryby mogą być także pomijane na podstawie wzajemnej relacji trybów użytych w sąsiadujących ze sobą blokach. Doświadczenie pokazuje, że podjęte wcześniej decyzje co do sposobu zakodowania bloków w znacznym stopniu determinują wybory trybów w blokach, które bezpośrednio sąsiadują z tymi, które już zakodowaliśmy. Jest to więc inny sposób na zawężenie puli sprawdzanych trybów.

Zamiast wykluczać z procedury testowania wybrane tryby (według przesłanek wskazanych powyżej), można opierać wybór trybów na wynikach analizy kodowanej treści lub tego, co już było. Może to być analiza złożoności tekstury fragmentu obrazu (w przypadku kodowania INTRA), czy weryfikacja decyzji, jakie podjął koder w poprzednio zakodowanych obrazach (kodowanie INTER). Już choćby prosta analiza charakteru kodowanej tekstury pomoże odpowiedzieć na pytanie w ramach jak dużych bloków najlepiej będzie kodowaną treść przewidywać (w toku kodowania INTRA). Tutaj sprawdzi się zasada: prosta treść – duże bloki, skomplikowana treść – podział na małe bloki. Dodatkowo, rezultat tej analizy może również wskazać, jaka jest kierunkowość danej tekstury, tzn. pod jakim kątem biegną zawarte w niej krawędzie. A dzięki takiej wiedzy szybko uda się wskazać numer najlepszego predyktora INTRA.

Podobna analiza złożoności treści raczej się nie sprawdzi w przypadku kodowania z predykcją międzyobrazową próbek. Tutaj jak wiemy, wysoką skuteczność przewidywania treści udaje się często uzyskać nie tylko w przypadku zastosowania małych bloków, ale również tych dużych. Ale to nie znaczy, że nic nie da się zrobić. Znaczna część treści sekwencji powtarza się w kolejnych obrazach. Kodując więc dany fragment obrazu możemy odwołać się do obrazu (bądź obrazów) które już wcześniej zakodowano, odszukać treść, która „pasuje” do tej którą aktualnie kodujemy i zobaczyć jak w tamtym przypadku postąpił koder. W ten sposób koder znakomicie zredukuje paletę sprawdzanych trybów. W tym podejściu można nawet zrezygnować z szukania „pasującej” treści – jeśli założy się, że z obrazu na obraz ruch obiektów sceny jest niewielki, to dojdziemy do wniosku, że wystarczy odwołanie do tej samej lokalizacji w zakodowanym obrazie

co położenie w obrazie aktualnie kodowanego fragmentu. Takie rozwiązanie uchodzi jednak raczej za skrajne.

W praktyce więc, uwzględniając aspekt złożoności obliczeniowej metody wyboru trybów, nie testuje się wszystkich możliwych kombinacji wariantów kompresji. Żeby utrzymać tę złożoność w rozsądnych granicach, określone tryby kodera trzeba zawczasu wykluczyć i w ogóle ich nie sprawdzać. Rzeczywiste, szybkie kodery obrazów robią to stosując pewną kombinację wspomnianych powyżej rozwiązań.

Jednak na samym wyborze trybów kompresji sterowanie koderem jeszcze się nie kończy. Trzeba jeszcze zadecydować o sile kwantowania danych resztkowych. I w tym miejscu dochodzimy do swoistego problemu „kury i jajka”. Obraz kodujemy przy uwzględnieniu założonych wcześniej ograniczeń na prędkość bitową zakodowanych danych (docelowa prędkość bitowa jest albo konkretnie określona jako wartość, albo zakłada się pewien dopuszczalny zakres wahań jej wartości). Celem sterowania koderem jest więc taki wybór trybów kodowania bloków, który przy spełnieniu kryteriów prędkości bitowej zmaksymalizuje jakość zakodowanych obrazów. Na prędkość bitową zakodowanych danych możemy w największym stopniu wpływać regulując szerokość kroku kwantyzacji w kwantyzatorze. Dlatego w praktyce koder ustawia kwantyzator jeszcze przed wyborem trybów kodowania. Jednak najlepsze ustawienie kwantyzatora zależy od tego, jakie są wartości danych, które chcemy skwantować. A te wynikają z kolei z zastosowanych trybów kodowania treści. Tak więc, dopóki nie wybierzemy trybów to nie wiemy tak naprawdę jakie ustawienia kwantyzatora najlepiej jest przyjąć. W praktyce problem ten jest rozwiązywany poprzez użycie modeli matematycznych, czyli specjalnych wzorów, które dla danego kodera pokazują związek pomiędzy prędkością bitową zakodowanych danych a przyjętą szerokością kroku kwantyzacji $\boldsymbol{Q_s}$. Wzory takie wyprowadza się w toku szeroko zakrojonych eksperymentów, w trakcie których bada się związek pomiędzy wynikami kodowania a przyjętymi ustawieniami kodera. Przykładowo, w przypadku kodera AVC relację tę można dla obrazu modelować funkcją:

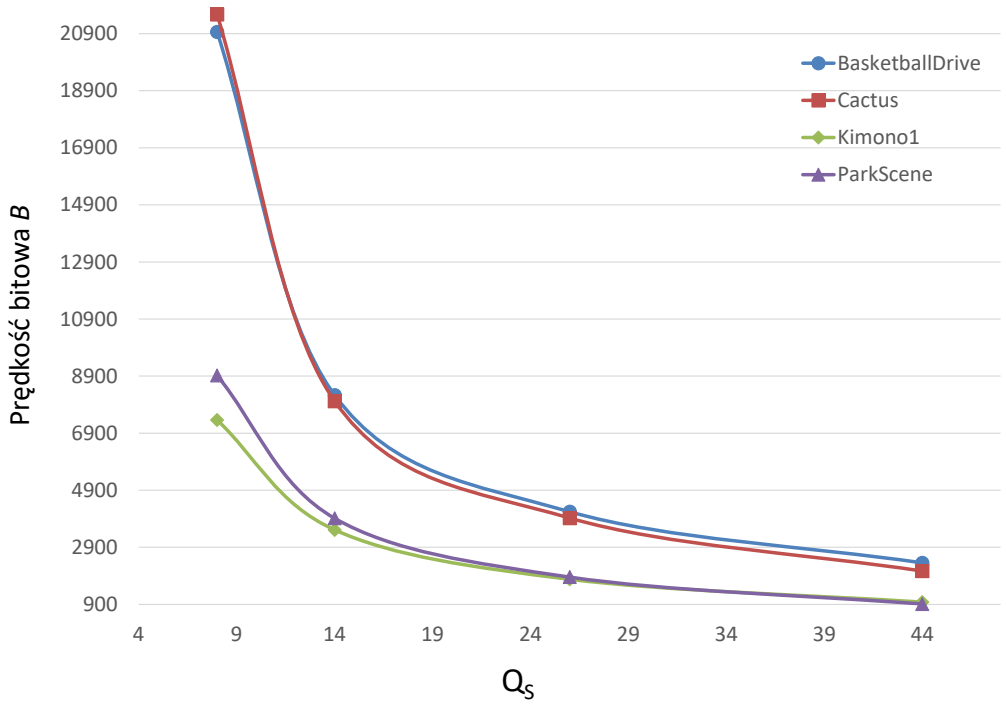
$$B - B_H = a \cdot \frac{MAD}{Q_s} + b \cdot \frac{MAD}{Q_s^2} \tag{4.5}$$

gdzie:
B – to docelowa liczba bitów, jaką przeznaczyliśmy na zakodowanie obrazu;
B_H – szacowana liczba bitów nagłówkowych i tych, które opisują wektory ruchu;
a, b – parametry modelu, które silnie zależą od treści obrazu;
MAD – średnia wartość bezwzględnych różnic (ang. mean absolute difference) pomiędzy wartościami próbek obrazu oryginalnego a wartościami próbek obrazu po kompresji i zdekodowaniu.

W oparciu o przyjęty wzór, wiedząc ile bitów chcemy poświęcić na zakodowanie bieżącego obrazu (czy fragmentu obrazu), dobiera się w pierwszej kolejności właściwy krok kwantowania $\boldsymbol{Q_s}$. I dopiero znając $\boldsymbol{Q_s}$, w kroku drugim, poszukuje się najlepszego zestawu trybów kodowania, które możliwie najlepiej (maksymalizacja jakości materiału po kompresji, przy spełnieniu założeń odnoszących się do kosztu bitowego) ten fragment (czy cały obraz) zakodują.

Tak działający blok sterowania koderem nie wydaje się aż tak bardzo skomplikowany. Jednak w rzeczywistości tak niestety nie jest. Można to sobie uzmysłowić analizując dokładniej elementy wzoru 4.5. Proszę zauważyć, że po pierwsze, na przedstawioną zależność $\boldsymbol{B}$-$\boldsymbol{Q_S}$ wpływ mają dodatkowo parametry $\boldsymbol{a}$ i $\boldsymbol{b}$, które zależą od kodowanej treści. Dla różnych sekwencji testowych krzywe $\boldsymbol{B}$-$\boldsymbol{Q_S}$ mogą się naprawdę znacząco między sobą różnić (patrz rysunek 4-78). Po drugie, żeby dla zadanej prędkości bitowej $\boldsymbol{B}$ dobrać najlepszy sposób kwantowania ($\boldsymbol{Q_s}$) to musimy z góry wiedzieć, jaki przyjąć poziom zniekształceń kompresji (miara $\boldsymbol{MAD}$), a ten jak wiemy

poznamy dopiero po przeprowadzaniu kompresji treści. Inaczej mówiąc, miara $\boldsymbol{MAD}$ zależy od $\boldsymbol{Q_s}$. Dochodzimy więc w tym miejscu do kolejnego problemu „kury i jajka” – w powyższym wzorze nie wiemy jaka jest wartość $\boldsymbol{MAD}$ bo ta zależy od zastosowanych trybów oraz $\boldsymbol{Q_s}$, i nie wiemy jakie zastosować $\boldsymbol{Q_s}$ bo ono zależy od $\boldsymbol{MAD}$. W praktyce więc uwzględnienie wymienionych czynników znacznie zwiększa trudność sterowania pracą kodera. Wyjściem z tego impasu jest przewidywanie wartości $\boldsymbol{a, b}$ oraz $\boldsymbol{MAD}$ na podstawie już zakodowanych bloków czy obrazów i zastosowanie tak oszacowanych wartości podczas kompresji następnych danych.



Rysunek 4-78. Zależność prędkości bitowej B od kroku kwantowania Q_s. Wykresy prezentują rzeczywiste wyniki, jakie z użyciem oprogramowania JM 18.2 kodera AVC zostały uzyskane dla wskazanych sekwencji testowych.

4.39. Przykłady wybranych wariantów sterowania koderem obrazu

4.39.1. Kodowanie ze stałym Qs

W tym przypadku w każdym z bloków obrazu przyjmuje się tę samą, wcześniej określoną wartość szerokości kroku kwantyzacji $\boldsymbol{Q_s}$. Ponieważ w obrazie możemy w ogólności napotkać bloki z „łatwą” oraz „trudną” do zakodowania treścią, to przy takim kodowaniu, z bloku na blok, może mocno nam się zmieniać liczba wytwarzanych przez koder bitów. Dodatkowo, jakość kolejnych fragmentów obrazu po kompresji może być w takim przypadku znacznie różna, czego negatywną konsekwencją mogą być zauważalne, silne wahania jakości zakodowanej treści. Wybór stałego $\boldsymbol{Q_s}$ w ogóle nie uwzględnia charakteru treści bloków, które są poddawane kompresji, dlatego ten sposób sterowania koderem jest bardzo nieefektywny. Można powiedzieć, że poza właściwymi badaniami w zakresie koderów obrazów, ten sposób sterowania nie znajduje merytorycznego uzasadnienia do jego zastosowania w praktyce.

4.39.2. Kodowanie w wariancie CBR

Ten wariant sterowania nazywany jest **trybem stałej prędkości bitowej** (ang. constant bitrate – **CBR**). Parametrem wejściowym sterowania jest wartość docelowej prędkości bitowej. W tym trybie sterowania koder dba o to, żeby np. w ramach każdego obrazu nie wyprodukować podczas kompresji więcej bitów niż zadeklarowana wartość. W niektórych przypadkach takie kodowanie może okazać się więc nieefektywne. W przypadku kodowania obrazów o skomplikowanej treści, deklarowana liczba bitów może okazać się niewystarczająca, żeby zakodować obraz z oczekiwaną jakością. Z kolei kodując proste obrazy liczba zużywanych bitów może się okazać niepotrzebnie zbyt wysoka. Taki sposób sterowania koderem może więc prowadzić do niewydajnego wykorzystania zasobów budżetu bitowego. Dlatego nie jest wskazane wykorzystywanie w praktyce tego wariantu sterowania koderem.

4.39.3. Kodowanie w wariancie ABR

Ten typ sterowania jest trybem stałej, **średniej prędkości bitowej** (ang. average bitrate – **ABR**). W tym przypadku chcemy, żeby średnia prędkość bitowa zakodowanych danych, mierzona w ustalonych odcinkach czasu, nie przekraczała zadanej wartości. Ponieważ prędkość tę będziemy na przykład mierzyć co sekundę, to daje to jakieś pole manewru do zastosowania innej (możliwie najlepszej) alokacji liczby bitów pomiędzy kolejne obrazy sekwencji. Możliwości dostosowania wyboru trybów do przyjętego budżetu bitowego są więc w tym przypadku większe niż w klasycznym wariancie **CBR** (patrz poprzedni punkt). Z uwagi na względną prostotę realizacji, oraz znacznie lepsze rezultaty w porównaniu z **CBR**, wariant **ABR** znajduje praktyczne zastosowanie. Chociaż ciągle nie jest to najwydajniejszy sposób sterowania koderem.

4.39.4. Kodowanie VBR

W literaturze ten sposób sterowania jest nazywany **kodowaniem ze zmienną prędkością bitową** (ang. variable bitrate – **VBR**). Kodowanie jest prowadzone tak, żeby uzyskać znacznie lepszą niż w wariancie **CBR**, relację koszt bitowy zakodowanych danych – jakość materiału po kompresji. Wyższą niż dla **CBR** efektywność kodowania danych uzyskuje się poprzez zastosowanie w **VBR** zaawansowanej, zależnej od treści dystrybucji bitów pomiędzy kolejnymi obrazami sekwencji oraz pomiędzy poszczególnymi fragmentami obrazu, wewnątrz każdego z obrazów. Dlatego w tym wariancie sterowania koderem sposób przydziału bitów pomiędzy kodowane fragmenty jest rzeczywiście kluczowy.

Oczywistym wyznacznikiem tego, ile bitów powinno się w trakcie kodowania poświęcić jest stopień skomplikowania kodowanego fragmentu (rodzaj tekstury, charakter ruchu). Na fragmenty prostsze można przeznaczyć mniej bitów, czego nie można zrobić w przypadku fragmentów bardziej złożonych (bo nie uda się ich wtedy dobrze zakodować). To, czy dany fragment jest złożony czy też nie określa się w koderze analizując w pewien sposób stopień złożoności tekstury bloków oraz rodzaj i dynamikę ruchu obiektów w sekwencji. Można powiedzieć, że ta prosta reguła alokacji bitów stanowi podstawę działania wszystkich koderów z trybem **VBR**.

Jednak powyższa przesłanka wydajnej alokacji bitów nie jest jedyna. Mniej oczywistym, ale również trudniejszym rozwiązaniem jest wykorzystanie ponadto wniosków z obserwacji preferencji widza. Na przykład, centralna część obrazu ma dla widza generalnie większe znaczenie niż brzegi obrazu. Większa uwaga widza skupia się na poruszających się obiektach sceny niż na treści obrazu,

która stanowi tło dla tych obiektów. Obiekty znajdujące się bliżej kamery mają większe znaczenie (z punktu widzenia percepcji treści) niż te, których odległość jest większa. I tak dalej. Tego typu przesłanki mogą stanowić znakomitą podstawę do oceny **percepcyjnej istotności treści** znajdującej się w poszczególnych fragmentach obrazu. W ostatnich latach zostały opracowane metody, które w zupełnie automatyczny sposób pozwalają przeanalizować treść obrazu pod kątem wspomnianej istotności percepcyjnej. Wynikiem działania tych metod są tzw. **mapy istotności** (ang. saliency maps), które wskazują poziom „ważności” poszczególnych fragmentów obrazu, oceniając to właśnie z perspektywy widza (patrz przykład na rysunku 4-79). Co ciekawe, rezultaty działania przywołanych metod dobrze korespondują z rzeczywistymi preferencjami widza (preferencjami w zakresie tego, które fragmenty obrazu są ważne, a które mniej). Dlatego w ostatnich latach zaczęły dochodzić do głosu metody sterowania, które wykorzystują w sposób łączny wymienione w tym punkcie podejścia.

Znając ograniczenia na dostępny budżet bitowy koder wybiera następnie możliwie najlepsze tryby kodowania i parametry kwantowania danych.

obraz

mapa istotności treści obrazu

Rysunek 4-79. Obraz wejściowy (u góry) oraz mapa percepcyjnej istotności treści (na dole) wejściowego obrazu. Na obrazie dolnym poziomem jasności zaznaczono istotność (ważność) danego fragmentu obrazu, z punktu widzenia percepcji obrazu przez widza. W oznaczeniu tym przyjęto zasadę: większa istotność treści – większa jasność fragmentu.

4.39.5. Wariant sterowania z optymalizacją R-D

4.39.5.1. Krótkie wprowadzenie

Szczególnym rodzajem sterowania koderem jest tzw. **optymalizacja R-D** (ang. rate-distortion optimization – **RD**). Celem tej optymalizacji nie jest osiągnięcie konkretnej, wcześniej zadanej, prędkości bitowej zakodowanego strumienia czy uzyskanie określonej jakości materiału po zakodowaniu. Tutaj algorytm sterowania podejmuje próbę jednoczesnej optymalizacji jednego i drugiego parametru, czyli i prędkości bitowej i jakości obrazów po kompresji. W tym wariancie sterowania stawiamy wobec tego przed koderem następujące zadanie: „Zaproponuj tryby kodowania bloków, plus sposób kwantowania danych tak, żeby otrzymać możliwie najlepszą jakość treści po zakodowaniu. Zwróć jednak szczególną uwagę na to, żeby uzyskany poziom jakości zakodowanych bloków warty był swojej ceny, wyrażonej liczbą bitów, jaka jest wytwarzana podczas kompresji bloków”.

Mówiąc inaczej, kodujemy treść w taki sposób, żeby wynikowa jakość zakodowanych obrazów faktycznie uzasadniała ponoszony bitowy koszt reprezentacji danych. Jeśli koder rozważa użycie jeszcze innych, bitowo bardziej kosztownych wariantów kompresji, to może to zrobić tylko wtedy, jeśli otrzymana na tej drodze dodatkowa poprawa jakości obrazów będzie faktycznie zauważalna dla widza. Ten dodatkowy wzrost jakości ma być warty swojej zwiększonej ceny reprezentacji danych (większy koszt bitowy).

Ten typ sterowania koderem jest zatem bardzo trudny. Wymaga każdorazowego (bo w przypadku każdego wariantu kodowania) badania poziomu degradacji jakości obrazów (w wyniku stratnego kodowania) i zliczania liczby bitów, które pojawiają się na wyjściu kodera. Żeby zrobić to dokładnie, to już na etapie wyboru trybów, należy przechodzić pełną ścieżkę kodowania treści danym zestawem trybów, włącznie z zastosowaniem kompresji entropijnej.

4.39.5.2. Optymalizacja R-D w ujęciu matematycznym

Żeby znaleźć najlepszy balans pomiędzy jakością zakodowanego materiału a liczbą wytworzonych bitów, definiuje się funkcję celu $\boldsymbol{J}$, która łączy w sobie miarę $\boldsymbol{D}$ poziomu zniekształceń w zakodowanym obrazie oraz wskaźnik liczby bitów $\boldsymbol{R}$, jakie w drodze kodowania obrazu generuje koder. Wspomniane parametry $\boldsymbol{D}$ i $\boldsymbol{R}$ są „łączone” w funkcji $\boldsymbol{J}$ poprzez dodatkowy parametr $\boldsymbol{\lambda}$, który jest nazywany mnożnikiem Lagrange’a. Funkcja $\boldsymbol{J}$ przyjmuje więc postać jak w równaniu 4.6, a celem kodera jest takie wysterowanie wyborem trybów i sposobem kwantowania danych, żeby tę funkcję zminimalizować (w matematyce ten sposób optymalizacji nazywany jest metodą mnożników Lagrange’a).

$$\min \boldsymbol{J} = \boldsymbol{D} + \lambda \cdot \boldsymbol{R} \tag{4.6}$$

W praktyce jednak nie da się takiej optymalizacji przeprowadzić na poziomie całego obrazu. Z punktu widzenia obliczeniowego byłoby to wprost niemożliwe. Dlatego wyboru najlepszego wariantu kodowania (czyli minimalizacji $\boldsymbol{J}$) dokonuje się na poziomie bloków obrazu. Optymalizacja działania kodera jest zatem robiona niezależnie w kolejnych blokach. Nasza funkcja

celu (nazywana również funkcją kosztu przyjętego sposobu kodowania bloku) przyjmuje w tej sytuacji postać:

$$\min \boldsymbol{J_{blok}} = \boldsymbol{D_{blok}}(\text{tryby}, \mathrm{Q}_S) + \lambda \cdot \boldsymbol{R_{blok}}(\text{tryby}, \mathrm{Q}_S) \tag{4.7}$$

Z tej postaci dobrze już widać, że zarówno miara $\boldsymbol{D_{blok}}$ jak również wskaźnik $\boldsymbol{R_{blok}}$ zależą od przyjętych trybów kodowania bloku, a także sposobu kwantowania danych (parametr Q_S). A sama funkcja kosztu $\boldsymbol{J_{blok}}$ zależy od tych właśnie wskaźników, i dodatkowo od parametru $\boldsymbol{\lambda}$.

Żeby w drodze powyższej optymalizacji udało się faktycznie znaleźć najlepszy kompromis pomiędzy $\boldsymbol{D_{blok}}$ i $\boldsymbol{R_{blok}}$ to trzeba zastosować właściwą wartość parametru $\boldsymbol{\lambda}$. Parametr ten jest swego rodzaju „negocjatorem” dwóch przeciwnych stron: poziomu zniekształceń w zakodowanej treści i kosztu bitowego jej reprezentacji. Proszę zauważyć, że w przypadku zastosowania dużej wartości $\boldsymbol{\lambda}$ wskaźnik $\boldsymbol{D_{blok}}$ byłby w powyższym wzorze spychany na margines, i wtedy algorytm optymalizacji dążyłby do zminimalizowania wielkości $\boldsymbol{R_{blok}}$, w ogóle nie zważając na poziom wprowadzanych zniekształceń $\boldsymbol{D_{blok}}$. Z kolei, zastosowaniu małej $\boldsymbol{\lambda}$ towarzyszyłby skutek odwrotny – algorytm dążyłby do minimalizacji $\boldsymbol{D_{blok}}$ nie zważając na bitowy koszt $\boldsymbol{R_{blok}}$. Nam zależy na „zbalansowaniu” wartości jednego i drugiego wskaźnika, stąd konieczność zastosowania właściwej wartości $\boldsymbol{\lambda}$. Tę właściwą wartość można wyznaczyć stosując znane w zagadnieniach minimalizacji, reguły matematyki.

Koniecznym warunkiem istnienia ekstremum funkcji $\boldsymbol{J_{blok}}$ (również ekstremum, które jest minimum tej funkcji) jest zerowanie się jej pierwszej pochodnej. Wyznaczając pochodną tej funkcji względem wielkości $\boldsymbol{R_{blok}}$ dochodzimy do następującego, ogólnego wzoru na $\boldsymbol{\lambda}$.

$$\frac{d\boldsymbol{J_{blok}}}{d\boldsymbol{R_{blok}}} = 0 = \frac{d\boldsymbol{D_{blok}}}{d\boldsymbol{R_{blok}}} + \lambda \quad \Rightarrow \quad \lambda = -\frac{d\boldsymbol{D_{blok}}}{d\boldsymbol{R_{blok}}} \tag{4.8}$$

Widać z niego, że parametr $\boldsymbol{\lambda}$ nie jest stałą wartością, ale zależy od relacji $\boldsymbol{D_{blok}}$ i $\boldsymbol{R_{blok}}$, które to wartości zależą z kolei od użytych trybów kodowania bloku, przyjętego sposobu kwantowania danych (parametr Q_S), oraz od treści kodowanego bloku. Dokładne uwzględnienie wszystkich tych czynników jest jednak w praktyce ekstremalnie trudne. Wiązałoby to się ze zbyt ogromnym nakładem obliczeń. Z tego powodu zadanie optymalizacji R- D poddaje się uproszczeniom. I tak po pierwsze, zależność $\boldsymbol{R_{blok}}$ - $\boldsymbol{D_{blok}}$ przybliża się funkcją 4.9.

$$\boldsymbol{R_{blok}}(\boldsymbol{D_{blok}}) = a \cdot log_2\left(\frac{b}{\boldsymbol{D_{blok}}}\right) \;\Rightarrow\; \boldsymbol{D_{blok}}(\boldsymbol{R_{blok}}) = b \cdot 2^{-\frac{\boldsymbol{R_{blok}}}{a}} \tag{4.9}$$

Pochodna tej drugiej funkcji jest taka jak przedstawiono to poniżej, z czego wynika już bardziej szczegółowa postać parametru $\boldsymbol{\lambda}$. Widać, że zależy on od treści, którą kodujemy (parametry a i b) oraz od kosztu bitowego $\boldsymbol{R_{blok}}$ kompresji bloku.

$$\frac{d\boldsymbol{D_{blok}}}{d\boldsymbol{R_{blok}}} = b \cdot \left(-\frac{1}{a}\right) \cdot 2^{-\frac{\boldsymbol{R_{blok}}}{a}} \cdot \ln 2 \quad \text{i} \quad \lambda = -\frac{d\boldsymbol{D_{blok}}}{d\boldsymbol{R_{blok}}} \;\Rightarrow\; \lambda = \frac{b}{a} \cdot 2^{-\frac{\boldsymbol{R_{blok}}}{a}} \cdot \ln 2 \tag{4.10}$$

I wreszcie po drugie, zależność $\boldsymbol{D_{blok}}$ i $\boldsymbol{Q_S}$ opisuje się wzorem:

$$\boldsymbol{D_{blok}(Q_s)} = \frac{Q_S^2}{3} \tag{4.11}$$

Przytoczone funkcje są wynikiem analizy obszernego zbioru danych eksperymentalnych, prezentujących rezultaty działania rzeczywistych koderów obrazu.

Łącząc ze sobą wzory na $\boldsymbol{R_{blok}(D_{blok})}$ oraz $\boldsymbol{\lambda(R_{blok})}$ otrzymuje się wyrażenie 4.12, które uzależnia wartość parametru λ od szerokości przedziału kwantyzacji Q_S.

$$\lambda = \frac{\ln 2}{3a} \cdot Q_S^2 \tag{4.12}$$

W powyższym wyrażeniu mamy jeszcze parametr a, na którego wartość wpływ ma treść bloku. Żeby nie wyznaczać już tego parametru za każdym razem (byłoby to trudne i algorytmicznie i obliczeniowo), przyjmuje się w koderach następujący, uproszczony wzór na $\boldsymbol{\lambda}$:

$$\lambda = 0{,}85 \cdot Q_S^2 \tag{4.13}$$

Powyższy sposób wyznaczenia parametru $\boldsymbol{\lambda}$ został przyjęty w algorytmach sterowania koderami H.263 i AVC.

Należy jednak zdawać sobie sprawę z tego, iż pomimo zastosowaniu szeregu uproszczeń w prezentowanym sposobie sterowania koderem obrazu, optymalizacja **R-D** ciągle cechuje się bardzo wysokim nakładem obliczeń.

Część IV – Rozwój technik kompresji obrazu z perspektywy ostatnich 30 lat

4.40. Wprowadzenie

Metody kompresji, które w największym stopniu wyznaczyły kierunek dla sposobu zapisu i transmisji obrazu (w tym także ruchomego obrazu) to **metody kodowania hybrydowego**. Kodery hybrydowe doczekały się wielu rynkowych wdrożeń, przez co ich praktyczne wykorzystanie jest obecnie bardzo szerokie. Z tego też powodu to właśnie te techniki kompresji obrazu były przedmiotem szczegółowych rozważań w tej książce.

Jednak przedstawione tutaj możliwości technik kompresji hybrydowej nie pojawiły się ot tak, z dnia na dzień. Są one rezultatem wielu lat badań, ulepszeń i modyfikacji znanych rozwiązań, ale także, sukcesywnego wprowadzania zupełnie nowych metod, których próżno było szukać w starszych koderach. W efekcie tego osiągnięto rzeczywiście ogromny postęp, którego kolejne odsłony znalazły swój wyraz w coraz lepszych koderach obrazu kolejnych generacji. Z punktu widzenia tego jak dzisiaj opisywany jest obraz (szczególnie ruchomy obraz) kluczowe są osiągnięcia ostatnich 2-3 dekad. W tym czasie światło dzienne ujrzały trzy generacje koderów, których przedstawicielami są dobrze w świecie znane techniki kodowania: **MPEG-2** (rok 1994), **AVC/H.264** (rok 2003) i **HEVC/H.265** (rok 2013).

Jak wielki postęp dokonał się w tym czasie? Jak to wpłynęło na osiągi kompresji oraz inne cechy techniki kodowania, jak np. złożoność obliczeniową, czy wielkość oprogramowania kodera

i dekodera obrazu? Jak zmieniły się poszczególne bloki funkcjonalne kodeka? W kolejnych punktach książka podejmuje próbę wskazania odpowiedzi na postawione wyżej pytania.

4.41. Ewolucja techniki kodowania hybrydowego

Można powiedzieć, że główna idea, zgodnie z którą działa hybrydowy koder obrazu nie zmieniła się na przestrzeni ostatnich 20-30 lat. Kodowaniu podlegają fragmenty obrazu (bloki). Poszczególnych fragmentów nie koduje się wprost, ale koder stara się najpierw przewidzieć ich treść na podstawie innych danych, którymi dysponują koder i dekoder. Jak już wiadomo, to przewidywanie nie jest jednak idealne, bezbłędne, dlatego w jego toku jest popełniany pewien błąd (błąd predykcji próbek bloku). I dopiero ten błąd jest w dalszym etapie przetwarzania danych przedmiotem kodowania transformatowego w koderze obrazu. Przedstawiona idea jest podstawą działania zarówno koderów starszych (np. koder MPEG-2 z 1994 roku), jak i tych najnowszych (np. kodery AVC – rok 2003, czy HEVC – rok 2013). Dlatego z tej perspektywy mogłoby się wydawać, że w sposobie działania kodera nic lub prawie nic, się nie zmieniło. Że wszystko jest „po staremu”.

Gdyby spojrzeć na wymienione kodery z „lotu ptaka”, to występujące między nimi różnice mogą być faktycznie słabo widoczne. Jednak jak to często bywa „diabeł tkwi w szczegółach”. Dlatego chcąc dokładnie porównać ze sobą dwa różne kodery obrazu należy zajrzeć do ich „wnętrza” i bezpośrednio porównać ze sobą stosowane w nich narzędzia kompresji danych. Czyli trzeba dokonać porównania na znacznie niższym poziomie. Jeśli porównania dokona się w taki sposób, to na jaw wyjdą ogromne wręcz różnice w działaniu wskazanych koderów. Mówiąc najogólniej różnice te będą dotyczyć trzech fundamentalnych kwestii:

1. Realizowanego przez koder sposobu podziału obrazu na bloki, w których jest realizowana kompresja danych.

2. Łącznej liczby wszystkich trybów kodowania, którymi dysponuje koder obrazu.

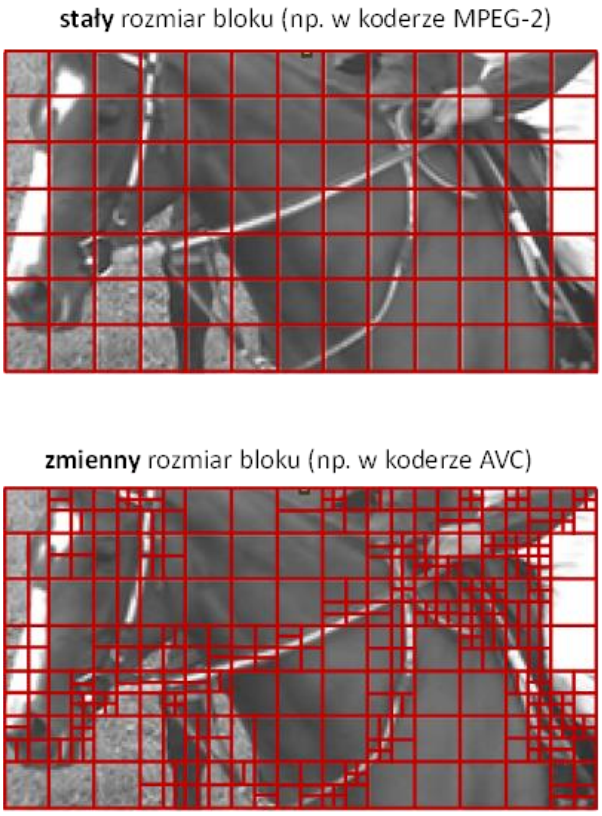
3. Stosowanych przez koder rozwiązań w zakresie poprawy jakości obrazu po kompresji.

4.41.1. Podział obrazu na bloki

Starsze kodery, jak MPEG-2 stosowały tylko jeden, wcześniej ustalony rozmiar bloku. Rozmiar ten był jednak inny dla etapów przewidywania wartości próbek (bloki o rozmiarze 16x16) oraz kodowania transformatowego próbek, bądź sygnału błędu predykcji tych próbek (bloki o wielkości 8x8). Samo przewidywanie wartości kodowanych próbek było zarezerwowane jedynie dla trybów kompresji międzyobrazowej (czyli brak przewidywania w kodowaniu wewnątrzobrazowym), co stanowiło dość znaczące uproszczenie działania kodera i dekodera obrazu. Zastosowanie stałego rozmiaru bloków odbierało starszym koderom zdolność adaptacji sposobu kodowania treści do jej charakteru (patrz rysunek 4-80). W sposób oczywisty więc, taki sposób podziału obrazu na bloki był powodem uzyskiwania przez koder słabszych wyników kompresji w niektórych fragmentach obrazu.

Dużo bardziej zaawansowane rozwiązania pojawiły się dopiero w późniejszych koderach obrazu, jak AVC, czy HEVC. Tutaj, oprócz bloków 16x16 (predykcja wartości próbek) oraz bloków 8x8 (kodowanie transformatowe sygnału) zaczęto stosować bloki o zupełnie nowych rozmiarach. I tak, w koderze AVC dodane zostały bloki 16x8, 8x16, 8x8, 8x4, 4x8, 4x4, na potrzeby

predykcji danych, oraz 4x4, dla potrzeb kodowania transformatowego. W tym zakresie koder HEVC poszedł jeszcze dalej. W stosunku do tego, co jest już w koderze AVC, w koderze HEVC mamy jeszcze bloki 64x64, 32x32, 16x16, 8x8, z dalszą możliwością ich podziału na dwa równe bloki (podział w kierunku pionowym lub poziomym), na równe cztery bloki, a z wyłączeniem bloków 8x8 również z niesymetrycznym podziałem bloku na dwa mniejsze (w kierunku pionowym lub poziomym), w proporcjach ¼ i ¾. Tak jest w przypadku predykcji próbek. W przypadku kodowania transformatowego koder HEVC może użyć bloków 4x4, 8x8, 16x16, 32x32. Nie ma tutaj wątpliwości, że wprowadzenie szerokiej palety dostępnych rozmiarów bloków w najnowszych koderach otworzyło drzwi do podziału obrazu nierównomierną siatką bloków, która jest dostosowana do treści kodowanego obrazu. Ten zabieg zaowocował znacznie trafniejszym przewidywaniem treści bloków oraz bardziej wydajnym kodowaniem transformatowym danych, co przekłada się finalnie na znacznie wyższą efektywność kompresji obrazu.



Rysunek 4-80. Stały oraz zmienny rozmiar bloków obrazu, stosowany odpowiednio w koderach starszych (np. MPEG-2) oraz nowszych (np. AVC, czy HEVC).

4.41.2. Liczba dostępnych trybów kompresji

Ubogi wachlarz rozmiarów bloków, jaki mają w swojej dyspozycji starsze kodery przekłada się na bardzo małą liczbę dostępnych wariantów kompresji treści. Niewielkie możliwości wyboru trybów są dobrze widoczne, tak w przypadku kompresji wewnątrzobrazowej, jak i kodowania międzyobrazowego.

Kodowanie wewnątrzobrazowe jest w starszych koderach bardzo proste. W tym trybie kompresji koder rezygnuje nawet z próby przewidzenia treści kodowanych bloków, czyli właściwe kodowanie transformatowe danych dotyczy nie sygnału błędu przewidywania wartości próbek, ale

bezpośrednio próbek obrazu. W oczywisty sposób przekłada się to na słabsze wyniki kompresji danych, w końcu samo kodowanie transformatowe nie jest tak efektywne jak zestawienie tego kodowania z predykcją wartości próbek. Tak daleko idącego uproszczenia nie znajdziemy już w przypadku kodowania międzyobrazowego. Jego główną siłą jest predykcja treści bloków na podstawie tego, co można znaleźć w obrazach z innych chwil czasowych. Tak więc, nawet w przypadku koderów starszych, taką predykcję (w kompresji międzyobrazowej) się faktycznie wykonuje. Jednak robi się to tylko w blokach o wielkości 16x16 (nie ma bloków ani mniejszych ani większych), co ogranicza dokładność przewidywania próbek. W związku z tym faktem również i w tym trybie kodowania bloków liczba możliwych wariantów kompresji jest silnie ograniczona.

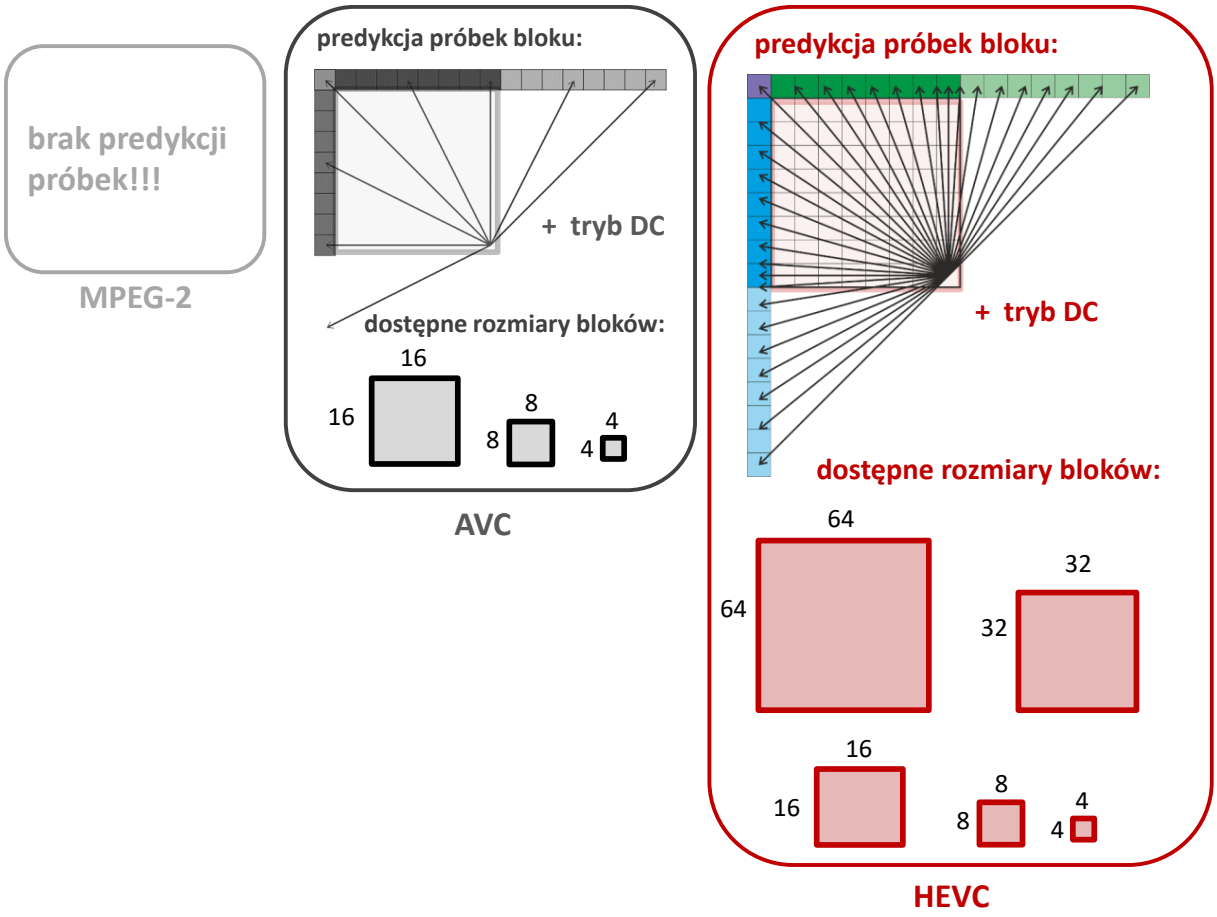
W toku rozwoju koderów obrazu ich działanie zostało jednak znacząco skomplikowane. Można to zauważyć porównując koder MPEG-2 z nowszymi od niego koderami AVC i HEVC. W kompresji wewnątrzobrazowej ostatnich dwóch koderów kodowanie transformatowe danych poprzedza już predykcja próbek bloków. W koderze AVC taką predykcję można przeprowadzić w blokach 16x16, 8x8 i 4x4, a w koderze HEVC, oprócz wymienionych rozmiarów, dochodzą jeszcze bloki 32x32 i 64x64[45]. W każdym ze wskazanych rozmiarów bloków kodery mogą przewidzieć ich treść na wiele różnych sposobów, przy czym w koderze AVC tych sposobów (predyktorów) jest maksymalnie 9, a w koderze HEVC aż 35. Najnowsze kodery decydują więc w jak dużych blokach przewidywać próbki, i z wykorzystaniem którego z dostępnych predyktorów najlepiej jest to zrobić. Sumarycznie daje to więc ogromną liczbę możliwych kombinacji zakodowania fragmentu obrazu, np. o wielkości 64x64 próbki. Zwiększa to możliwości kompresji danych, ale z drugiej strony, znakomicie utrudnia wybór najlepszego trybu.

Nie inaczej jest w przypadku kompresji międzyobrazowej. Paleta rozmiarów bloków, w których można takie kodowanie przeprowadzić znacząco się zwiększyła w kolejnych generacjach koderów obrazu. Jest to wzrost od jednego dostępnego bloku w koderze MPEG-2 (16x16 – w przypadku kodowania sekwencji bez przeplotu), poprzez bloki 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4 w koderze AVC, do zestawu rozmiarów bloków 64x64, 32x32, 16x16, 8x8, z możliwością dalszego podziału tych bloków na dwa bloki (podział w pionie lub w poziomie na dwa bloki równe lub analogicznie, ale z podziałem na niesymetryczne bloki[46]) lub cztery jednakowe, mniejsze bloki w koderze HEVC. Podobnie jak w przypadku wewnątrzobrazowej kompresji daje to ogromną liczbę możliwych sposobów zakodowania fragmentu obrazu.

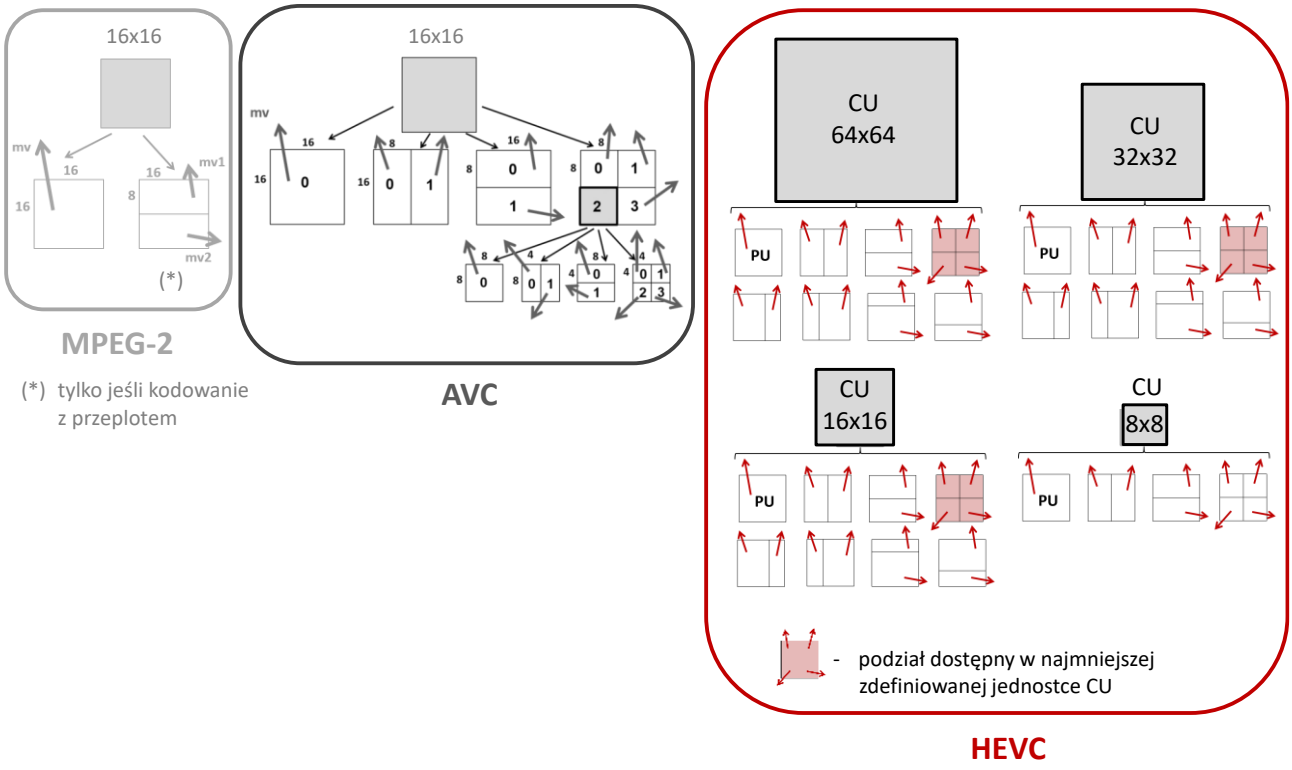
W tym miejscu należy jeszcze wspomnieć o zupełnie nowych, specjalnych trybach kompresji treści, których kodery starsze w ogóle nie stosowały. Mowa o takich trybach kodowania jak **SKIP** i **DIRECT**. Te tryby pozwalają na ekstremalnie wydajną kompresję fragmentów obrazu, dla których trafnie udało się przewidzieć dane o ruchu w sekwencji oraz treść kodowanego bloku (czyli po kwantyzacji danych błąd przewidywania wartości próbek jest zerowy) – tryb SKIP lub samą tylko informację o ruchu – tryb DIRECT.

[45] W koderze HEVC, predykcja treści bloku o wielkości 64x64 przebiega trochę nietypowo. Zamiast dokonywać predykcji całego takiego bloku, to blok ten dzieli się na 4 mniejsze bloki o rozmiarze 32x32 i przewiduje się treść tych mniejszych bloków. Jednak dla każdego z tych 4 bloków stosuje się ten sam predyktor wartości próbek, przez co tryb predykcji jest sygnalizowany tylko raz (na poziomie bloku 64x64). Proszę zauważyć, że jest to inny wariant predykcji treści niż ten, w którym w każdym z bloków 32x32 stosuje się odmienny predyktor treści (co wymaga sygnalizacji trybu cztery razy, bo na poziomie bloków 32x32).

[46] Podział na niesymetryczne bloki nie jest dozwolony w bloku 8x8.



Rysunek 4-81. Porównanie mechanizmów wewnątrzobrazowej predykcji próbek bloków w trzech koderach: MPEG-2, AVC i HEVC.



Rysunek 4-82. Porównanie mechanizmów międzyobrazowej predykcji próbek bloków w trzech koderach: MPEG-2, AVC i HEVC.

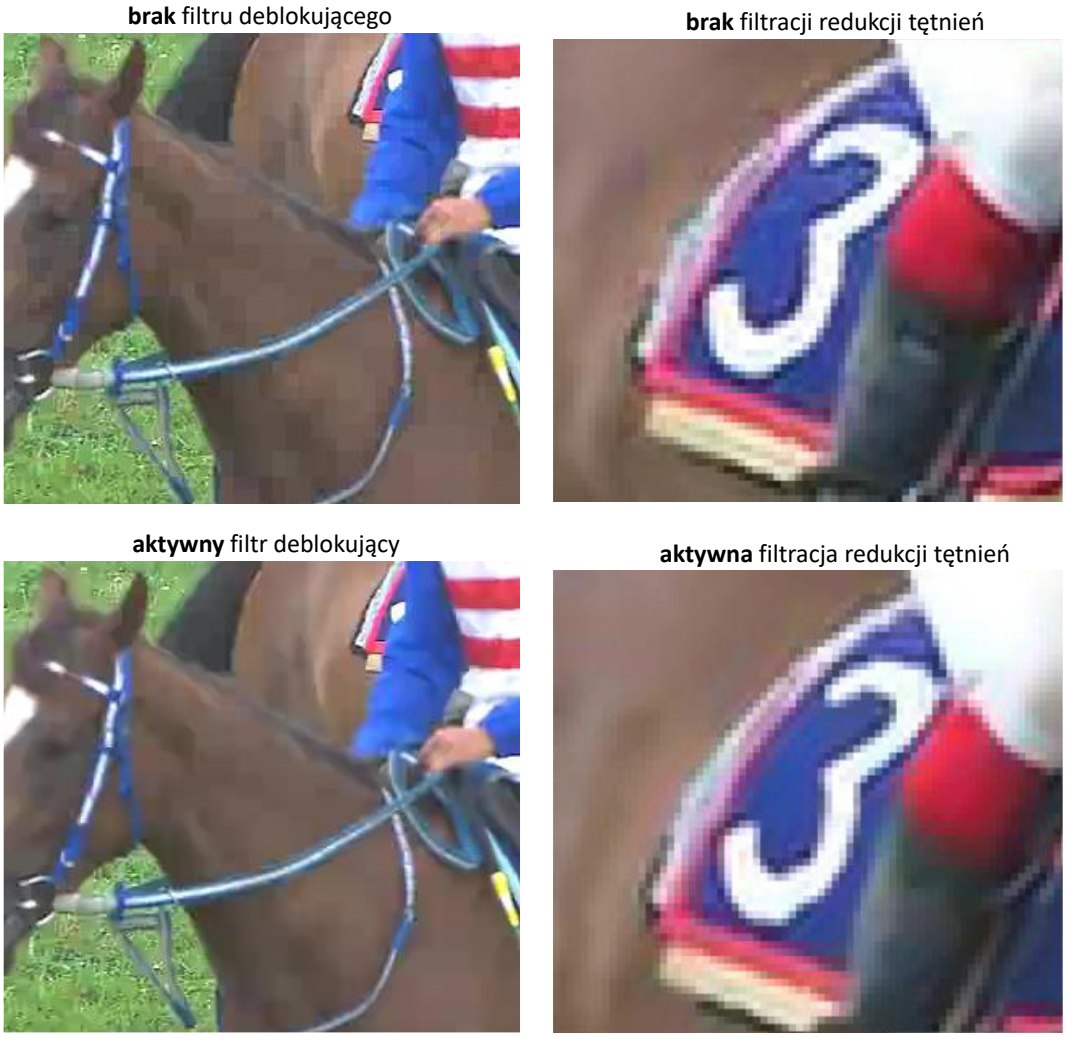
4.41.3. Poprawa jakości obrazów po kompresji

W wyniku kwantowania amplitud kosinusów (kosinusy te stanowią częstotliwościową reprezentację sygnału błędu predykcji próbek bloków obrazu, bądź bezpośrednio częstotliwościową reprezentację próbek bloków) w obrazie może się pojawić szereg zniekształceń treści. Najbardziej znamienne są dwa zniekształcenia: tzw. **efekt blokowy** oraz **efekt dzwonienia** na krawędziach (nazywany też **efektem tętnień**).

Powodem obu zniekształceń jest kodowanie stratne sygnału (stratne, bo określone dane są poddawane kwantyzacji), które koder wykonuje niezależnie od siebie w kolejnych blokach obrazu. Na skutek takiego kodowania, w obrazie zrekonstruowanym, może dojść do sytuacji, kiedy to na granicy dwóch sąsiednich bloków wartości próbek obrazu znacząco się między sobą różnią. W obrazie pojawiają się wtedy charakterystyczne kwadraty, właśnie na skutek pojawienia się pozornej krawędzi pomiędzy sąsiednimi blokami obrazu. Ten typ zniekształcenia jest właśnie efektem blokowym. Dodatkowo, kwantowanie danych, które opisują obraz utrudnia, czy wręcz całkowicie uniemożliwia, poprawne odtworzenie niektórych elementów treści obrazu. W szczególności dotyczy to krawędzi w obrazie, czyli elementów treści o szybkozmiennym charakterze, ponieważ koder zwykle silniej kwantuje właśnie składowe wysokoczęstotliwościowe obrazu. Po kwantyzacji, z powodu braku tych składowych (lub ich reprezentowania, ale bardzo zgrubnego), wokół krawędzi obrazu pojawiają się charakterystyczne tętnienia, zafalowania, które stanowią efekt dzwonienia na krawędziach. Opisywane zniekształcenia treści można zobaczyć na zamieszczonych poniżej obrazach (patrz rysunek 4-83), przy czym pokazane „błędy” są tym większe im silniej w kwantyzatorze kwantuje się dane.

Z przywołanymi zniekształceniami kompresji stratnej wiążą się dwa negatywne skutki. Po pierwsze jakość obrazu po kompresji jest znacznie niższa niż oryginału. I po drugie, zniekształcenia kodowania stratnego obniżają dodatkowo efektywność kompresji obrazów. Tak jak pierwszy skutek jest oczywisty, to drugi może być już zastanawiający. Ale łatwo go wytłumaczyć. Po kompresji w obrazach pojawiają się zniekształcenia, których nie ma w oryginalnych obrazach. Obrazy przed kompresją i po kompresji zaczynają się więc znacznie między sobą różnić. Ponieważ koder i dekoder dokonują przewidywania treści na podstawie tego, co zostało już wcześniej zakodowane i zdekodowane, to wspomniane zniekształcenia kompresji potęgują błędy predykcji próbek obrazu. Stąd właśnie niższa efektywność kompresji danych. Jednak pomimo wskazanych bolączek starsze kodery obrazu, jak np. MPEG-2, nie starały się wskazanych zniekształceń treści wyeliminować, czy chociażby w jakimś stopniu zredukować.

Jednak takie działania podejmują już nowsze kodery obrazu, np. AVC czy HEVC. Żeby widz miał wrażenie lepszej jakości obrazu, i dodatkowo, żeby poprawić skuteczność przewidywania treści na etapie samej kompresji, to w pętli sprzężenia zwrotnego kodowania i dekodowania obrazów stosuje się filtr lub filtry poprawiające jakość obrazów. W przypadku kodera AVC jest to tylko jeden filtr, nazywany **deblokującym**, który znacząco ogranicza występowanie efektu blokowego. W koderze HEVC, oprócz tego filtru, jest jeszcze drugi, który w zauważalny sposób redukuje efekt dzwonienia na krawędziach obiektów obrazu. Użycie każdego z tych filtrów z osobna zwiększa efektywność kompresji obrazów o kilka procent, jednak odbywa się to za cenę zwiększonej złożoności kodera i dekodera obrazów.



Rysunek 4-83. Ilustracja efektu blokowego oraz efektu dzwonienia (tętnień) w obrazie (dwa obrazy w pierwszym wierszu). Obrazy w drugim wierszu prezentują faktyczne możliwości redukcji wymienionych typów zniekształceń treści. Zaleca się oglądanie zamieszczonych obrazów w powiększeniu, celem łatwiejszego zauważenia błędów kompresji oraz efektów filtracji obrazów.

4.42. Ewolucja efektywności kompresji obrazów

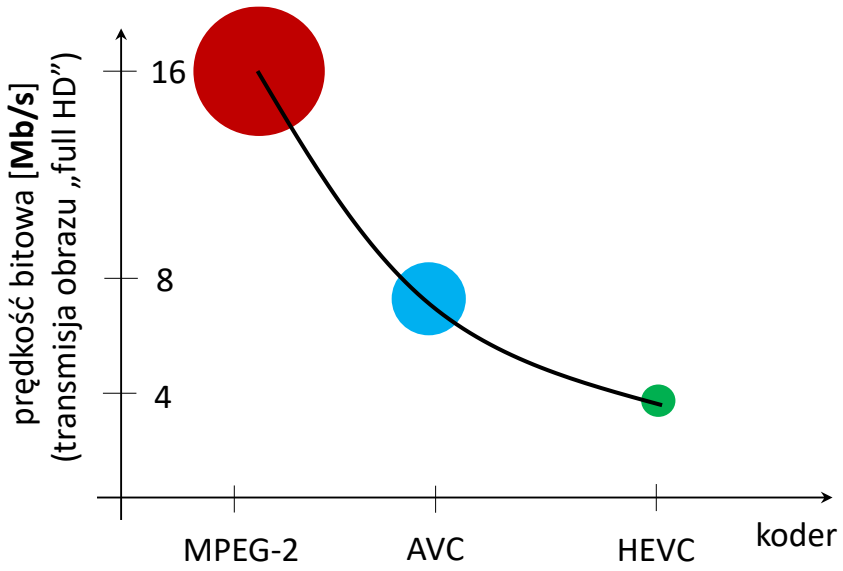
Z punktu widzenia efektywności kompresji danych kodery starsze (jak np. MPEG-2), oraz te najnowsze (np. HEVC) dzieli po prostu przepaść. W toku samych tylko 20 lat badań nad kompresją udało się zmniejszyć objętość danych ruchomych obrazów aż **4 krotnie**, nie pogarszając przy tym jakości tych obrazów po zakodowaniu! W kontekście kompresji danych obrazowych jest to osiągnięcie bardzo znaczące.

Legendarny koder **MPEG-2** (opracowany na początku lat 90-tych), pracujący z obrazami o wysokiej rozdzielczości przestrzennej (1920x1080 próbek w obrazie, czyli obrazy full HD) pozwalał na transmisję tych obrazów w dobrej jakości przy prędkości strumienia bitowego na poziomie **16 Mb/s** (zakładając kodowanie i transmisję 30 obrazów na sekundę). Z perspektywy transmisji takiego obrazu w sieciach telewizji kablowej było to ciągle dość dużo. Jednak po blisko 10 latach badań nad ulepszeniami metod kompresji udało się tę prędkość zmniejszyć aż **2-krotnie**! Zwieńczeniem tych prac była nowa technika kompresji **AVC/H.264**, której pierwsze wydanie

pojawiło się w roku 2003. W przypadku tego kodera wystarczyło już **8 Mb/s**, żeby osiągnąć tę samą jakość zakodowanych obrazów, w przypadku której koder **MPEG-2** potrzebował **16 Mb/s**. Tak znaczący postęp w dziedzinie kompresji umożliwił znacznie szersze wykorzystanie telewizji cyfrowej, w tym transmisję obrazów o wyższej rozdzielczości, co po roku 2003 rzeczywiście stało się faktem.

Jednak sukces techniki AVC nie zatrzymał starań nad jeszcze lepszymi efektami kompresji. Po roku 2003 takie prace trwały nadal. Uwzględniając rosnące wymagania użytkowników zmienił się obszar zastosowań nowych koderów. Otóż, oprócz dotychczasowego materiału, celem stała się również wydajna kompresja obrazów o wysokiej i ultra wysokiej rozdzielczości (przestrzenna rozdzielczość obrazów odpowiednio 1920x1080 i 3840x2160 próbek). Kolejne 10 lat wysiłków doprowadziły nas w roku 2013 do nowej techniki **HEVC** (ang. high efficiency video coding) kompresji obrazów. Ta technika okazała się być **2 razy** wydajniejsza niż **AVC** i aż **4 razy** bardziej wydajna niż legendarny koder **MPEG-2**! Zastosowanie nowej techniki dało więc możliwość transmisji obrazów full HD już przy prędkościach bitowych na poziomie **4 Mb/s**, zapewniając przy tym wysoką (lub co najmniej dobrą) jakość obrazów po kompresji. Proszę zauważyć, że w stosunku do objętości danych nieskompresowanych jest to zmniejszenie ilości danych przeszło **350-krotne**! A i tak widz nie widzi różnicy jakości pomiędzy obrazem zakodowanym a jego oryginalną wersją. Tak wydajna kompresja obrazów otwiera już możliwość szerokiego wdrożenia systemów multimedialnych nowej generacji, w których przesyła się obrazy o rozdzielczościach nawet wyższych niż full HD. Takie systemy, z transmisją obrazów ultra HD, są obecnie przedmiotem rynkowych wdrożeń (rok 2018).

Podsumowaniem tego punktu jest rysunek 4-84, na którym raz jeszcze pokazano, jak wielki postęp w zakresie wydajności kodowania obrazów udało się osiągnąć na przestrzeni ostatnich 20-30 lat.



Rysunek 4-84. Efektywność kompresji trzech koderów wizyjnych: MPEG-2, AVC i HEVC. Dotyczy kodowania 30 obrazów w czasie trwania jednej sekundy.

4.43. Jak zmieniła się złożoność kodowania obrazów?

Na przestrzeni lat, oprócz samej efektywności kompresji obrazów, istotnie zmieniła się również złożoność kodowania i dekodowania obrazów. Nastąpił dramatyczny wzrost złożoności. Można powiedzieć, że jest to cena, jaką przychodzi nam płacić za dokonany postęp.

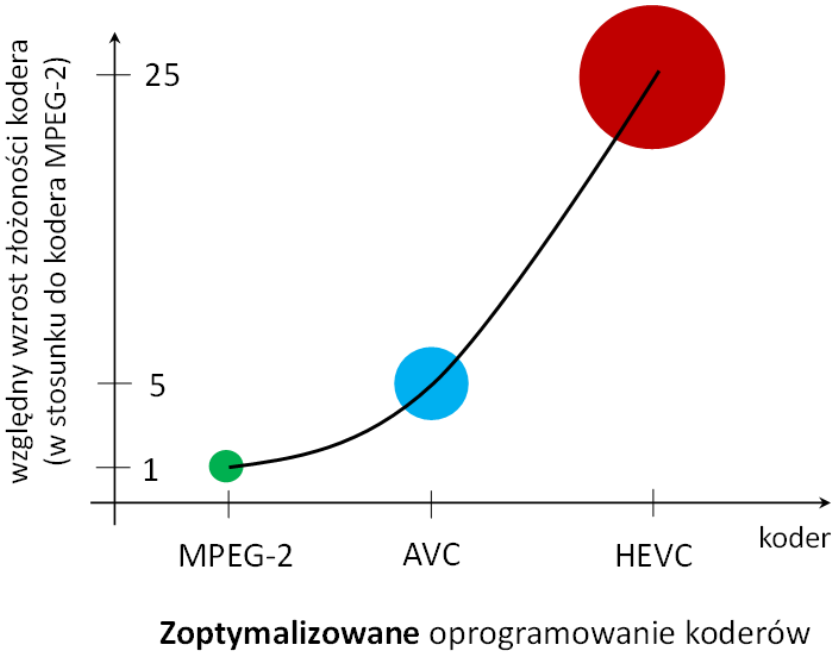
Powodem wzrostu złożoności jest zastosowanie w kolejnych kodekach coraz bardziej wydajnych, ale równocześnie bardziej skomplikowanych, narzędzi kompresji treści. Taki stan rzeczy odnosi się do wszystkich elementów kodera i dekodera obrazu: od przewidywania treści bloków, po transformatowe kodowanie resztkowych danych predykcji próbek, aż po entropijne kodowanie danych. Na wzrost złożoności wpływają także nowe techniki przetwarzania sygnału, których nie stosowano w starszych kodekach, ale są już używane w kodekach najnowszych. Tak jest w przypadku filtracji, która poprawia jakość obrazów po kompresji.

Skoro już wiemy, że złożoność kodeków uległa na przestrzeni lat istotnej zmianie, to czas odpowiedzieć na pytanie jak duży jest to wzrost? Można się o tym łatwo przekonać porównując ze sobą parametry złożoności[47] trzech kolejnych generacji kodeków: **MPEG-2**, **AVC** i **HEVC**. W celu uzyskania wiarygodnych danych przedmiotem porównania będą zoptymalizowane na prędkość działania, programowe implementacje trzech wskazanych technik kompresji, które w sposób wydajny potrafią wykorzystać zasoby współczesnych procesorów x86. Poszczególne implementacje kodeków zawierają podobne techniki optymalizacji kodu programu.

Tak sporządzone oprogramowanie kodera **MPEG-2** pozwala na kompresję około **100 obrazów** full HD w czasie jednej sekundy, jeśli realizować takie kodowanie na jednym wątku procesora Core i7, o częstotliwości zegara 3,4 GHz. Mowa oczywiście o kodowaniu, w którym uzyskuje się dobrą lub bardzo dobrą jakość zakodowanych obrazów oraz wysoką (z perspektywy danej techniki kodowania) efektywność kompresji danych. Koder **AVC** zakoduje już takich obrazów około **20** w ciągu sekundy. Odpowiada to więc **5-krotnie** wyższej złożoności kodera **AVC**, w porównaniu z koderem **MPEG-2**. A nowy koder **HEVC** będzie kodował jeszcze dłużej niż **AVC**. Tutaj różnica złożoności będzie również **5-krotna**, co odpowiada możliwości kompresji zaledwie **3-4** obrazów na sekundę na rozważanej platformie. W stosunku do starego już kodera **MPEG-2** koder **HEVC** jest więc aż **25-krotnie** bardziej złożony! Przytoczone wyniki zostały dodatkowo przedstawione na wykresie 4-85. Wykres ten jasno pokazuje, że obserwowany w kolejnych latach przyrost złożoności technik kompresji obrazów ma wykładniczy charakter. W praktyce, mniej więcej co dekadę, przychodzi nam mierzyć się z nową techniką kompresji, której złożoność jest **kilka razy wyższa** od techniki poprzedniej.

Z postępem w zakresie możliwości kompresji danych wiąże się zatem istotny wzrost złożoności kodowania obrazów. Złożoność dekodera obrazu także się zauważalnie zwiększa. Wzrost złożoności jest oczywiście wynikiem użycia w koderze i dekoderze bardziej skomplikowanych, i przez to obliczeniowo znacznie bardziej wymagających, algorytmów kodowania i dekodowania. Jednak jest jeszcze inna przyczyna tego stanu rzeczy. Jest nią coraz większa liczba narzędzi kompresji, które pozostają w dyspozycji kodera i dekodera obrazu. Jeśli tych narzędzi jest mało, to szybko można sprawdzić efektywność każdego z nich i w krótkim czasie wybrać najlepszy wariant kompresji danego fragmentu obrazu. Tego ostatniego nie da się zrobić, jeśli narzędzi kompresji jest bardzo dużo, np. tak jak w koderze HEVC. Olbrzymia liczba kombinacji możliwych sposobów zakodowania bloków w najnowszych koderach silnie utrudnia wybór najlepszego trybu. Nie da się tego zrobić szybko. Nawet, jeśli określone tryby są w procedurze sprawdzania całkowicie pomijane, to jakąś ich część trzeba ostatecznie przetestować. Wraz ze wzrostem palety sprawdzanych trybów mocno rośnie czas kodowania obrazów.

[47] Złożoność obliczeniowa jest wyrażana czasem działania kodera lub dekodera obrazu na procesorze komputera.



Rysunek 4-85. Porównanie złożoności trzech koderów obrazu: MPEG-2, AVC i HEVC. Wyniki złożoności dla zoptymalizowanych, programowych realizacji koderów.

4.44. Realizacja oprogramowania kodeka – jak zmieniła się trudność?

Wraz ze wzrostem stopnia skomplikowania poszczególnych narzędzi kodera i dekodera obrazu mocno zwiększyła się trudność opracowania oprogramowania kodeka. Wzrost tej trudności wynika dodatkowo również z tego, że w nowszych kodekach liczba dostępnych narzędzi, które musi zrealizować programista jest nieporównywalnie większa niż w przypadku kodeków starszych. A zaimplementowanie każdego z narzędzi kompresji wymaga przecież czasu.

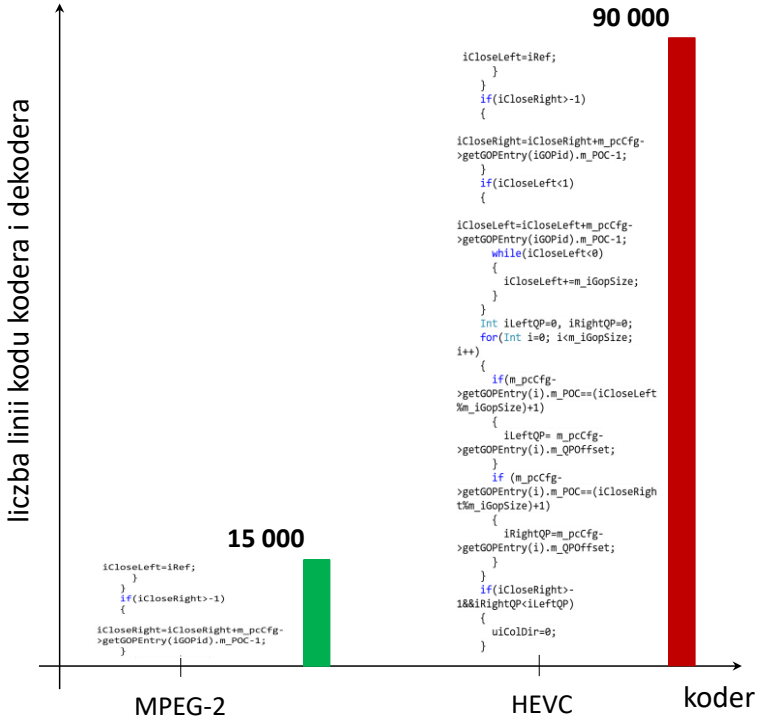
Powyższa kwestia w liczbach wygląda następująco. Programowa implementacja kodeka **MPEG-2** (koder + dekoder), wykonana w języku programowania C/C++ jest stosunkowo niewielka, ponieważ składa się na nią tylko około **15 tysięcy** linii kodu programu. Dla porównania, nowy kodek **HEVC** to już **90-100 tysięcy** linii kodu! Różnica wielkości oprogramowania jest więc bardzo, bardzo duża. Jest ona aż **6-7 krotna**! (patrz poniższy rysunek)

Jednak jak powyższa różnica przekłada się na realną czasochłonność wykonania kodu programu? Na pierwszy rzut oka mogłoby się wydawać, że o tyle, o ile większe jest oprogramowanie nowych kodeków, o tyle wydłuży się czas tworzenia jego implementacji, w porównaniu z kodekami starszymi. Jednak doświadczenie autora pokazuje, że tak optymistycznie się to niestety nie skaluje. Bardziej skomplikowane narzędzia kompresji znacznie trudniej się implementuje. Dodatkowo, w nowych kodekach takich narzędzi jest dużo. Jednego i drugiego nie można powiedzieć o kodekach starszych generacji. Żeby oprogramowanie nowego kodeka mogło korzystać z zasobów komputera w wydajny sposób, konieczna jest przemyślana i sprawna wymiana danych pomiędzy poszczególnymi komponentami w koderze i dekoderze. Jest to kolejny czynnik, o którym trzeba pomyśleć tworząc implementację. Wszystko to sprawia, że w praktyce czas opracowania kodu nowego kodeka (takiego jak **HEVC**) będzie co najmniej **10 krotnie** dłuższy niż starszego kodeka **MPEG-2**. Zdaniem autora wskazana krotność dobrze oddaje stan faktyczny, chociaż rzeczywista, dokładna wartość będzie również zależeć od szeregu różnych czynników. Wśród nich należy z pewnością wymienić to, jak realizujemy sterowanie pracą kodera (czyli jakie mechanizmy wyboru

trybów, czy kontroli prędkości bitowej strumienia zakodowanych danych implementujemy), oraz jaki przyjmujemy stopień i rodzaj optymalizacji kodu programu.

Jednak spróbujmy dodatkowo wskazać, jaki długi jest czas przygotowania oprogramowania kodera i dekodera. Tego starszego i tego nowego. Otóż, w przypadku kodeka **MPEG-2** można takie oprogramowanie wykonać już w ciągu **2-3 miesięcy**, gdyby zlecić to zadanie wykwalifikowanemu zespołowi **2-3 programistów**. W tej sytuacji, dodatkowego czasu wymagałyby prawdopodobnie dalsze pracy, których celem byłby kolejny etap optymalizacji kodu programu. W przypadku nowego kodeka **HEVC** ten sam zespół osób spędziłby nad zadaniem więcej niż **20 miesięcy**! Zatem czasy wykonania kodu nowych technik są bardzo długie. Wymagają więc niemałych nakładów finansowych i bardzo wyspecjalizowanej kadry. Z tego powodu mało jest na świecie zespołów (nie mówiąc już o naszym kraju), które tym zadaniem na co dzień się trudnią.

Należy przewidywać, że czas i trudność wykonania sprzętowych realizacji kodeków będą jeszcze większe niż realizacji programowych.



Rysunek 4-86. Porównanie wielkości oprogramowania kodeków MPEG-2 i HEVC. Różnica wielkości oprogramowania, wyrażona liczbą linii kodu programu, jest aż 6-krotna.

4.45. Jaka jest najbliższa przyszłość?

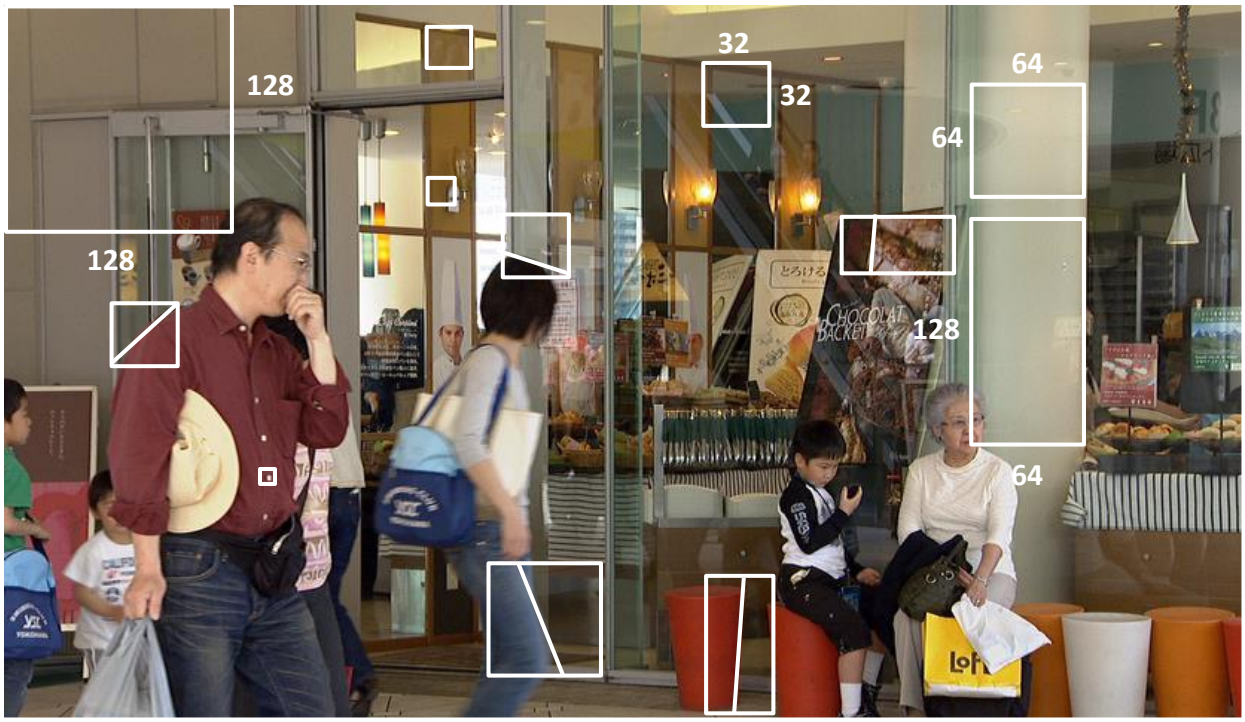
Doświadczenie ostatnich **20-30 lat** pokazuje, że co około **10 lat** pojawia się nowa technika kompresji obrazu, której efektywność jest aż **2-krotnie wyższa** niż techniki poprzedniej. Nie pogarszając więc jakości zakodowanego obrazu potrafimy go reprezentować, przeznaczając na ten cel coraz mniejsze nakłady bitowe. Pewne jest, że ten postęp jeszcze się nie zatrzymał.

Opracowana w 2013 roku technika **HEVC** kompresji obrazu nie kończy w żaden sposób prac, badań nad jeszcze wydajniejszymi metodami. Takie prace oczywiście się toczą. Bardzo dobrym tego przykładem jest obecna aktywność grup ekspertów **ISO/IEC MPEG** (ang. Moving

Picture Experts Group) oraz **ITU-T VCEG** (ang. Video Coding Experts Group), którą gremia te prowadzą od ostatnich kilku lat. Celem tych wysiłków jest opracowanie nowej techniki kompresji, nazwanej **VVC** (ang. Versatile Video Coding – VVC), która ma być następcą wdrażanej obecnie na rynku techniki **HEVC**. Finalizacja prac nad techniką **VVC** jest spodziewana na przełom lat 2020/2021, przy czym zakłada się, że do tego czasu uda się ponownie **2-krotnie** zredukować strumień bitów opisujący obrazy (w stosunku do możliwości kompresji, jakie daje technika **HEVC**), nie pogarszając przy tym jakości zakodowanych obrazów. Już dzisiaj znane są propozycje rozwiązań techniki **VVC**, które zwiększają efektywność kodera **HEVC** o około **40%**, co pokazuje, że założony cel jest jak najbardziej do osiągnięcia. Kwestią otwartą jest natomiast to, jak duży wzrost złożoności kodowania jesteśmy w stanie zaakceptować.

Prace nad przyszłą techniką VVC trwają w najlepsze. Dlatego dzisiaj (początek roku 2019) trudno jest jednoznacznie wyrokować, jakie narzędzia kompresji zostaną w **VVC** ostatecznie przyjęte. Wszystko natomiast wskazuje, że w dalszym ciągu będzie to kolejne, bardzo znaczące, ulepszenie przedstawionej w tej książce idei kompresji hybrydowej. Dlatego naturalnym punktem startowym dla prac nad **VVC** stały się rozwiązania techniki **HEVC** (oraz inne propozycje technik, które opracowano w ramach prac nad HEVC, ale ostatecznie nie zostały do HEVC włączone). Spośród aktualnie rozważanych ulepszeń kodera i dekodera są między innymi:

1. **Nowe rozmiary bloków** obrazu, i to na każdym z poziomów kodowania czy dekodowania: poziomie decyzji o stosowanym trybie kompresji – INTRA lub INTER, poziomie predykcji próbek bloku, i w końcu, poziomie transformatowego kodowania danych. Bloki będą mogły być w ogólności kwadratowe, jak i prostokątne. Oprócz tego rozważa się również możliwość „przecięcia” bloku pewną prostą, która biegnie pod zadanym kątem, co doprowadzi do otrzymania dwóch nowych bloków, których kształt jest inny niż kwadratowy czy prostokątny (takie rozwiązanie planuje się przyjąć na potrzeby predykcji próbek obrazu). Kierunek tych zmian został zilustrowany na rysunku 4-87.

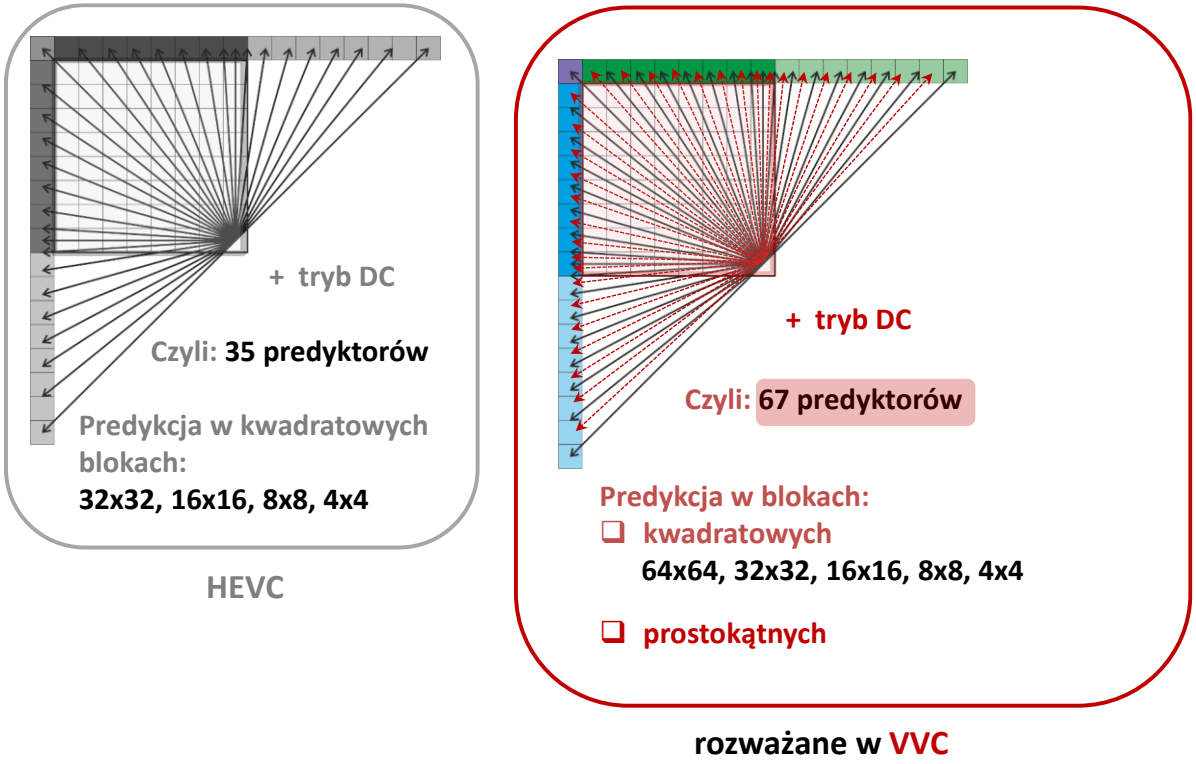


Rysunek 4-87. Rozważane w VVC rodzaje i rozmiary bloków. Wachlarz dostępnych rozmiarów bloków będzie znacznie większy niż w przypadku wcześniejszych koderów.

2. **Dokładniejsza wewnątrzobrazowa predykcja** treści (patrz poniższy rysunek). Większą precyzję przewidywania treści bloków planuje się uzyskać poprzez: 1) większą liczbę predyktorów treści bloków, oraz 2) szerszy wachlarz rozmiarów bloków, których treść będzie się przewidywać.

   W nowym koderze HEVC treść bloku można przewidywać z użyciem jednego z **35** dostępnych predyktorów. W opracowywanym koderze **VVC** predyktorów ma być znacznie więcej, bo aż przeszło **60**.

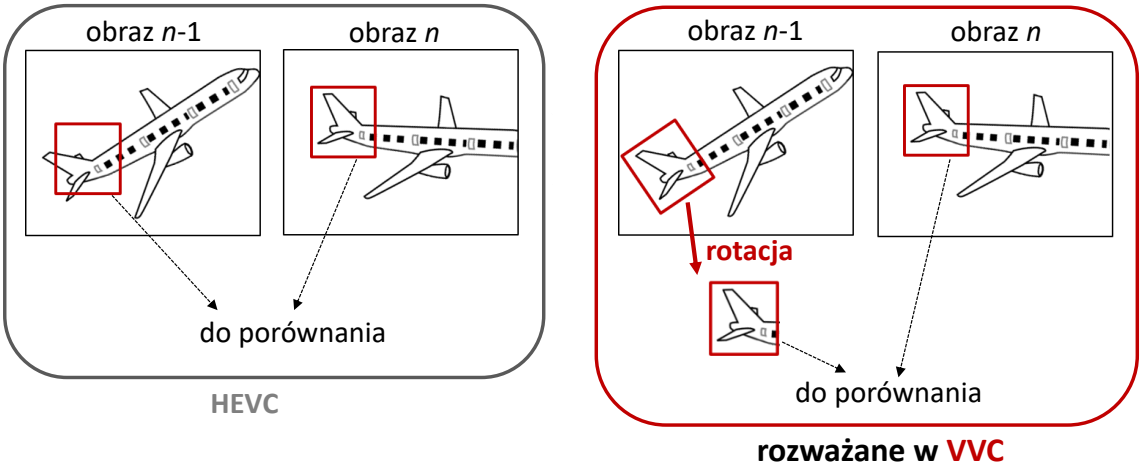
   Dodatkowo większy będzie wachlarz rozmiarów bloków, których treść będzie się przewidywało w drodze kodowania INTRA – oprócz wprowadzenia bloków o większym rozmiarze (niż bloki kodera **HEVC**), predykcję będzie można dodatkowo realizować w blokach o prostokątnym kształcie, o orientacji poziomej i pionowej (dla porównania, koder **HEVC** dokonuje predykcji próbek tylko w kwadratowych blokach).



Rysunek 4-88. Rozważane w VVC sposoby predykcji wewnątrzobrazowej próbek bloków. Wachlarz dostępnych predyktorów treści oraz rozmiarów bloków, których treść podlega przewidywaniu będzie znacznie większy niż w przypadku wcześniejszych koderów obrazu.

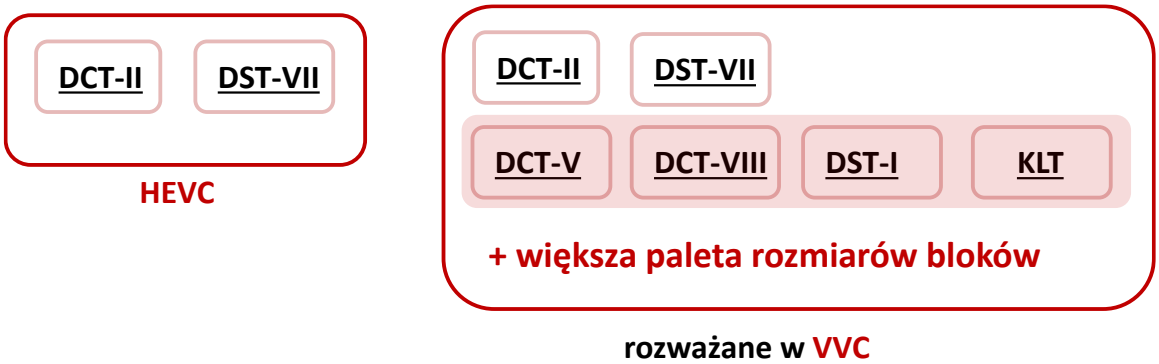
3. **Dokładniejsza międzyobrazowa predykcja** treści. Dotychczasowe kodery obrazu przewidują ruch obiektów w sekwencji zakładając, że sprowadza się on do zaledwie prostego, translacyjnego przemieszczania się bloków na płaszczyźnie obrazu. Jest to tzw. **model ruchu translacyjnego** obiektów sceny. Model ten w ogóle nie uwzględnia szeregu rodzajów ruchu i zmian treści, które rzeczywiście występują w kodowanych sekwencjach, jak na przykład przybliżenie/oddalenie, obrót fragmentu treści, czy inne nieregularne zmiany treści. Ponieważ uproszczony model ruchu znacząco ogranicza skuteczność przewidywania treści (przynajmniej w niektórych fragmentach obrazu), to w

opracowywanym aktualnie koderze **VVC** postanowiono to zmienić. Tutaj, estymację i kompensację ruchu będzie mogła poprzedzić **afiniczna transformacja** bloku próbek, która pozwoli na dużo dokładniejsze odwzorowanie ruchu obiektów (patrz poniższy przykład).



Rysunek 4-89. Przykład bardziej zaawansowanej estymacji ruchu obiektów sekwencji, która wykorzystuje obrót bloków. W pracach nad koderem VVC rozważa się nowe, bardziej skomplikowane modele ruchu, w których wykorzystuje się afiniczną transformację bloków obrazu.

4. **Bardziej zaawansowane kodowanie transformatowe i kodowanie entropijne**. Resztkowy sygnał predykcji próbek bloków planuje się kodować z wykorzystaniem większej liczby różnych transformacji, nawet w porównaniu z kodowaniem **HEVC**. Kierunek tych zmian został wskazany na poniższym rysunku.



DCT – dyskretne przekształcenie kosinusowe
DST – dyskretne przekształcenie sinusowe
KLT – przekształcenie Karhunena-Loevego

Rysunek 4-90. Rodzaje transformacji danych, jakie rozważa się w ramach przyszłej techniki VVC.

Entropijną kompresję danych planuje się realizować z wykorzystaniem nieco ulepszonej wersji kodowania CABAC.

Tak jak wcześniej już powiedziano, przedstawione ulepszenia kodera obrazu już dzisiaj zwiększają efektywność kompresji o około **40%** (w stosunku do kodowania **HEVC**), jednak ten przyrost jest okupiony przeszło **10-kotnym** zwiększeniem złożoności kodera (również w porównaniu do złożoności kodera **HEVC**) [Chen18a].

Pisząc o przyszłości metod kompresji nie sposób nie wspomnieć również o innej inicjatywie, rozpoczętej przez wielkie światowe koncerny. Mowa o technice kompresji obrazu, która ma być wolna od patentów, a tym samym dostępna bez licencyjnych opłat. Niestety, tą cechą nie mogą się pochwalić kodeki obrazu, które zostały opracowane przez grupę **MPEG** do spółki z **VCEG** (kodeki **MPEG-2**, **AVC** i **HEVC**). Właśnie ten czynnik szczególnie mocno hamuje dzisiaj wdrożenie rynkowe nowej techniki **HEVC**.

Dlatego realną alternatywą dla wyżej wymienionych kodeków jest nowa, wolna od patentów i opłat licencyjnych, technologia kodowania **AOMedia Video 1 (AV1)** [Chen18b], opracowana w ostatnim czasie przez Alliance for Open Media (**AOMedia**). **AOMedia** jest założonym w 2015 roku konsorcjum, które skupia w swoich szeregach duże firmy z sektora przemysłu półprzewodnikowego, dostawców usług „wideo na żądanie” oraz firmy z branży teleinformatycznej: Amazon, Apple, ARM, Cisco, Facebook, Google, IBM, Intel, Microsoft, Mozilla, Netflix, Nvidia, Adobe, Ateme, BBC R&D, Bitmovin, Hulu, i VideoLAN, oraz inne firmy. W toku prac konsorcjum, przyjmując jako punkt startowy technologię **VP9** firmy Google [Mukh13], została opracowana nowa technika **AV1**, o której mówi się, że efektywnością kodowania danych przewyższa kodery **VP9** i **HEVC** o **30%**. Większe są jednak nakłady obliczeniowe na kodowanie i dekodowanie danych. Podobnie jak inne omówione w tej książce techniki kompresji, **AV1** również realizuje kodowanie hybrydowe (przez co, można się zastanawiać, czy rzeczywiście metody tej techniki nie naruszają żadnego z istniejących patentów). Technika **AV1** nie będzie tutaj szerzej omawiana, natomiast można powiedzieć, że jej bloki funkcjonalne, chociaż nie takie same, to są jednak bardzo podobne to narzędzi koderów **HEVC** i **VVC**. Najbliższe lata na pewno upłyną pod znakiem zaciętej rywalizacji koderów **HEVC** i **VVC** oraz technologii **AOMedia**.

4.46. Podsumowanie rozdziału i wnioski końcowe

W ciągu ostatnich 20-30 lat dokonał się ogromny postęp w zakresie technik reprezentacji cyfrowych obrazów. Motorem dla rozwoju metod kompresji były ciągle zwiększająca się popularność mediów cyfrowych, rosnąca moc obliczeniowa układów scalonych i procesorów, postęp w zakresie wyświetlaczy i kamer, ale również, coraz większe wymagania widzów odnośnie jakości prezentowanego im obrazu. Pewien splot wymienionych czynników determinował stopień zaawansowania technik kompresji, które można było wdrożyć na rynek w danym czasie.

W toku lat badań nad kompresją naukowcy i inżynierowie zaproponowali wiele różnych podejść do reprezentacji obrazu [Doma10]. Jednak spośród nich wszystkich niewątpliwie największy sukces, również rynkowy, odniosły **techniki kodowania transformatowego** obrazu, w przypadku obrazu statycznego, a jeśli mowa o obrazie ruchomym, **techniki kompresji hybrydowej**. Dlatego uwaga tego rozdziału została skupiona na tych właśnie metodach kodowania. Ogromne powodzenie wymienionych technik jest wynikiem ich bardzo dobrych parametrów kompresji, umiarkowanej złożoności kodowania i dekodowania obrazu, ale również

szerokich możliwości dalszego ulepszania stosowanych algorytmów. Ta ostatnia cecha dała możliwość wprowadzenia na rynek kolejnych, coraz to wydajniejszych generacji hybrydowych i transformatowych koderów obrazu, z których wszystkie dzielą tę samą główną ideę kodowania danych. Dobrym tego przykładem są przedstawione w tej książce rozwiązania technik **MPEG-2**, **AVC** i **HEVC**, które uznać można za kamienie milowe przeszło 20-letniej historii rozwoju koderów obrazu. Wymienione kodery dobrze pokazują skalę postępu, jaki dokonał się w tym czasie w zakresie kompresji, kiedy to w przypadku kodowania ruchomego obrazu udało się zwiększyć stopień kompresji danych z około **90** (koder **MPEG-2**) do przeszło **350** (koder **HEVC**), przy zachowaniu ciągle wysokiej jakości zakodowanego materiału. Jak dobrze pokazuje teraźniejszość, ten burzliwy rozwój koderów jeszcze się nie zakończył, trwa nadal, czego dobrym przykładem jest nadchodząca wielkimi krokami technologia **VVC** (ang. Versatile Video Coding – **VVC**). Ta technika kompresji ma umożliwiać kompresję przeszło **600-700-krotną**, nie wprowadzając przy tym widocznych zniekształceń do kodowanych obrazów.

W książce skupiono się tylko na kompresji statycznych oraz ruchomych obrazów, które rejestruje się pojedynczym aparatem lub kamerą. Były to więc tradycyjne **obrazy monoskopowe**. Rozwój urządzeń rejestrujących, ale również technik kompresji danych stworzyły możliwość akwizycji oraz wydajnej reprezentacji również dużo bardziej skomplikowanych sygnałów, jak **obrazy stereoskopowe**, **obrazy trójwymiarowe**, tzw. **obrazy 360°** oraz **obrazy immersyjne**. Z tymi obrazami wiążą się znacznie większe objętości danych (w porównaniu z monoskopowymi obrazami), dlatego ich wydajna kompresja nabiera jeszcze większego znaczenia niż w przypadku klasycznego sygnału. I jest na pewno dużo trudniejsza. Kompresja nowych typów obrazu również korzysta z technik przedstawionych w tej książce, jednak dodatkowo eksploatuje także inne, dedykowane narzędzia kodowania danych, których w książce nie przytoczono.

W kolejnych latach należy się spodziewać dalszych ulepszeń i rozszerzeń idei kompresji hybrydowej i transformatowej, skutkujących jeszcze wyższą efektywnością kompresji obrazów, w stosunku do możliwości, jakie zostały zaprezentowane w tej książce.

Bibliografia

[Abhil17] A. Abhilash, G. Sreelekha, HEVC-based lossless intra coding for efficient still image compression, Multimedia Tools and Applications, January 2017, Vol. 76, Issue 2, pp. 1639-1658.

[Ahmed74] Ahmed, N., T. Natarajan, R. K. Rao, Discrete Cosine Transform, IEEE Transactions on Computers, C-23:90-93, 1974.

[Apos85] A. Apostolico, and A. S. Fraenkel, Robust transmissions of unbounded strings using Fibonaci representations. Technical Report CS85-14, Dpt of Applied Mathematics, The Weizmann Institute of Science, Rehovot Israel, 1985.

[Arai88] Arai Y., Agui T., Nakajima M., A fast DCT-SQ scheme for images, Transactions of the IEICE, vol. E-71, 1988, s. 1095-1097.

[AVC] ISO/IEC 14496-10, Generic Coding of Audio-Visual Objects, Part 10: Advanced Video Coding, March 2006.

[AVCSoft] H.264/AVC software coordination site – http://bs.hhi.de/~suehring/tml.

[AVS] Audio Video Coding Standard Workgroup of China (AVS), The Standards of People's Republic of China GB/T 20090.2-2006, Information Technology – Advanced Coding of Audio and Video – Part 2:Video, 2006.

[Asano15] Yuta Asano, Individual Colorimetric Observers for Personalized Color Imaging, PhD Dissertation, Rochester Institute of Technology, 2015.

[Begl04] R. Begleiter, R. El-Yaniv, and G. Yona, On Prediction Using Variable Order Markov Models. Journal of Artificial Intelligence Research, Vol. 22 (2004), pp. 385-421, December 2004.

[Bely06] E. Belyaev, M. Gilmutdinov, and A. Turlikov, Binary Arithmetic Coding System with Adaptive Probability Estimation by "Virtual Sliding Window". IEEE International Symposium on Consumer Electronics, pp. 1-5, 2006.

[Bjønt01] G. Bjøntegaard, Calculation of Average PSNR Differences between RD curves. ITU-T SG16/Q6, 13th VCEG Meeting, Austin, Texas, USA, April 2001, Doc. VCEG-M33.

[bzip2] J. Seward, Bzip2 and libbzip2, Version 1.0.3. A program and library for data compression. http://www.bzip.org.

[Chen18a] Huanbang Chen i inni, Description of SDR, HDR and 360° video coding technology proposal by Huawei, GoPro, HiSilicon, and Samsung, Document JVET-J0025, 10th Meeting: San Diego, US, 10–20 Apr. 2018.

[Chen18b] Chen, Y., Murherjee, D., Han, J., Grange, A., Xu, Y., Liu, Z., Parker, S., Chen, C., Su, H., Joshi, U., Chiang, C.-H., Wang, Y., Wilkins, P., Bankoski, J., Trudeau, L., Egge, N., Valin, J.-M., Davies, T., Midtskogen, S, Norkin, A., de Rivaz, P.: An overview of core

coding tools in the AV1 video codec. In: Picture Coding Symposium (PCS), 24–27 June 2018, San Francisco, California, United States (2018, submitted)

[Chen77] Wen-Hsiung Chen, C. Smith and S. Fralick, A Fast Computational Algorithm for the Discrete Cosine Transform, in IEEE Transactions on Communications, vol. 25, no. 9, pp. 1004-1009, Sep 1977. doi: 10.1109/TCOM.1977.1093941

[Clear84] J. G. Cleary, and I. H. Witten, Data Compression Using Adaptive Coding and Partial String Matching. IEEE Transactions on Communication, Vol. 32, No. 4, pp. 396-402, April 1984.

[Doma98] M. Domański, Zaawansowane techniki kompresji obrazów i sekwencji wizyjnych. Poznań, 1998.

[Doma10] M. Domański, Obraz cyfrowy. Wydawnictwa Komunikacji i Łączności, 2010.

[Elias75] P. Elias, Universal Codeword Sets and Representation of the Integers. IEEE Transactions on Information Theory, Vol. 21, No. 2, pp. 194-203, 1975.

[Firo06] M. H. Firooz, and M. S. Sadri, Improving H.264/AVC Entropy Coding Engine Using CTW method. Picture Coding Symposium, April 2006.

[FLAC] “FLAC - Free Lossless Audio Codec,” http://xiph.org/flac/, Oct. 2013.

[Gall75] R. G. Gallager, and D. Van Voorhis, Optimal Source Codes for Geometrically Distributed Integer Alphabets. IEEE Transactions on Information Theory, Vol. 21, pp. 228-230, March IT-1975.

[Gall78] R. G. Gallager, Variations on a Theme by Huffman. IEEE Transactions on Information Theory, Vol. IT-24, No. 6, pp. 668-674, 1978.

[Ghan04] M. Ghandi, M. M. Ghandi, and M. B. Shamsollahi, A Novel Context Modeling Scheme for Motion Vectors Context-Based Arithmetic Coding. IEEE Canadian Conference on Electrical & Computer Engineering (CCECE04), May 2-5, 2004, Ontario, Canada.

[Golo66] S. W. Golomb, Run-Length Encoding. IEEE Transactions on Information Theory, Vol. IT-12, pp. 399-401, 1966.

[gzip] GZIP Homepage, http://www.gzip.org.

[H263] ITU-T Rec. H.263, Video Coding for Low Bit Rate Communication, August 2005.

[HEVC] G. J. Sullivan, J. R. Ohm, W. J. Han, T. Wiegand, Overview of the High Efficiency Video Coding (HEVC) standard, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 22, No. 12, pp. 1649 – 1668, December 2012.

[HEVC rec] ISO/IEC and ITU-T, High Efficiency Video Coding (HEVC), ISO/IEC 23008-2 (MPEG-H Part 2) / ITU-T Rec. H.265, April 2013.

[HEVCbook] VivienneSze, Madhukar Budagavi, Gary J. Sullivan (Editors), High Efficiency Video Coding (HEVC). Algorithms and Architectures, Springer, 2014.

[HM] HEVC test model reference software – available online: https://hevc.hhi.fraunhofer.de

[Hong04] D. Hong, M. van der Schaar, and B. Pesquet – Popescu, Arithmetic Coding with Adaptive Context-Tree Weighting for the H.264 Video Coders. Visual Communications and Image Processing 2004, Vol. 5308, pp. 1226-1235, January 2004.

[Huff52] D. A. Huffman, A Method for the Construction of Minimum-Redundancy Codes. Proceedings of the I. R. E, pp. 1098-1101, September, 1952.

[ITU-R BT.709] ITU-R Rec. BT.709-5, Parameter values for the HDTV standards for production and international programme exchange, 2002.

[Jain81] J. R. Jain, and A. K. Jain, Displacement Measurement and Its Application in Interframe Image Coding. IEEE Trans. Commun. Vol. COM-29, pp. 1799-1808, 1981.

[JBIG] ISO/IEC 11544 and ITU-T Rec. T.82, Information Technology – Coded Representation of Pictures and Audio Information – Progressive Bi-Level Image Compression, March 1993.

[JBIG2] ISO/IEC 14492 and ITU-T Rec. T.88, JBIG2 Bi-Level Image Compression standard, 2000.

[JPEG] ISO/IEC 10918-1 and ITU-T Rec. T.81, Information Technology – Coded Representation of Picture and Audio Information – Digital Compression and Coding of Continuous-Tone Still Images (JPEG standard), 1993.

[JPEGLS] ISO/IEC 14495-1 and ITU-T Rec. T.87, Information Technology – Lossless and Near Lossless Compression of Continuous – Tone still Images, 1999.

[JPEG2000] ISO/IEC 15444-1 and ITU-T Rec. T.800, Information Technology – JPEG 2000 image coding system, 2000.

[Karw06] D. Karwowski, Ulepszone Adaptacyjne Kodowanie Arytmetyczne w Zaawansowanym Koderze Wizyjnym H.264/AVC Wykorzystujące Ważenie Drzew Kontekstów (CTW). V Sympozjum Naukowe “Techniki Przetwarzania Obrazu”, str. 469-475, Serock, Listopad 2006.

[Karw07a] D. Karwowski, Improved Arithmetic Coding in H.264/AVC Using Context-Tree Weighting and Prediction by Partial Matching. European Signal Processing Conf. EUSIPCO 2007, pp. 1270-1274, September 2007, Poznań, Poland.

[Karw07b] D. Karwowski, and M. Domański, Improved Arithmetic Coding In H.264/AVC Using Context-Tree Weighting Method. Picture Coding Symposium PCS 2007, November 2007, Lisboa, Portugal.

[Karw08] D. Karwowski, Advanced Adaptation Algorithms of Arithmetic Coding in Hybrid Video Compression. Doctoral Dissertation, Poznań University of Technology, 2008.

[Karw12] D. Karwowski, Significance of Entropy Coding in Contemporary Hybrid Video Encoders, Advances in Soft Computing no. 184, Image Processing and Communications Challenges 4 , wyd. Springer-Verlag, 2012, ss.111-117.

[Loeff89] C. Loeffler, A. Ligtenberg and G. S. Moschytz, "Practical fast 1-D DCT algorithms with 11 multiplications," *International Conference on Acoustics, Speech, and Signal Processing,*, Glasgow, 1989, pp. 988-991 vol.2. doi: 10.1109/ICASSP.1989.266596

[Mah05] M. Mahoney, Adaptive Weighting of Context Models for Lossless Data Compression. Florida Tech. Technical Report CS-2005-16, 2005.

[Marp03a] D. Marpe, H. Schwarz, and T. Wiegand, Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 7, pp. 620-636, July 2003.

[Marp03b] D. Marpe, and T. Wiegand, A highly Efficient Multiplication-Free Binary Arithmetic Coder and Its Application in Video Coding. IEEE International Conference on Image Processing (ICIP'03), Barcelona, Spain, September 2003.

[Marp04] D. Marpe, Adaptive Context-Based and Tree-Based Algorithms for Image Coding and Denoising. Doctoral Dissertation, Universität Rostock, Rostock 2004.

[Marp06a] D. Marpe, G. Marten, and H. L. Cycon, A Fast Renormalization Technique for H.264/MPEG4-AVC Arithmetic Coding. 51st Internationales Wissenschaftliches Kolloquium Technische Universitat Ilmenau, September 2006.

[Marp06b] D. Marpe, H. Kirchhoffer, and G. Marten, Fast Renormalization for H.264/MPEG4-AVC Arithmetic Coding. Proc. 14th European Signal Processing Conference (EUSIPCO 2006), Florence, Italy, September 4-8, 2006.

[Mila06] S. Milani, and G. A. Mian, An Improved Context Adaptive Binary Arithmetic Coder for the H.264/AVC Standard. Proc. of European Signal Processing Conference (EUSIPCO 2006), September 4-8, 2006, Florence, Italy.

[MPEG-1] ISO/IEC 11172-2, Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 Mbit/s, Part 2: Video. (MPEG-1), November 1993.

[MPEG-2] ISO/IEC 13818-2 and ITU-T Rec. H.262, Generic Coding of Moving Pictures and Associated Audio Information – Part 2: Video. (MPEG-2), November 1994.

[MP2AAC] ISO/IEC 13818-7, Information Technology – Generic Coding of Moving Pictures and Associated Audio, Part 7: Advanced Audio Coding, 1997.

[MP4AAC] ISO/IEC 14496-3, Information Technology – Coding of Audio Visual Objects – Audio, 2001.

[Mrak03a] M. Mrak, D. Marpe, and S. Grgic, Comparison of Context-Based Adaptive Binary Arithmetic Coders in Video Compression. Proc. EC-VIP-MC'03, July 2003.

[Mrak03b] M. Mrak, D. Marpe, and T. Wiegand, Application of Binary Context Trees in Video Compression. Picture Coding Symposium (PCS'03), St. Malo, France, April 2003.

[Mrak03c] M. Mrak, D. Marpe, and T. Wiegand, A Context Modeling Algorithm and its Application in Video Coding. IEEE International Conference on Image Processing (ICIP'03), Barcelona, Spain, September 2003.

[Mukh13] Mukherjee, D., Bankoski, J., Grange, A., Han, J., Koleszar, J., Wilkins, P., Xu, Y., Bultje, R.S.: The latest open-source video codec VP9 - an overview and preliminary results. In: Picture Coding Symposium (PCS), December 2013

[Pas76] R. Pasco, Source Coding Algorithms for Fast Data Compression. Doctoral Dissertation, Stanford University, 1976.

[Penn88] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, and R. B. Arps, An Overview of the Basic Principles of the Q-coder Adaptive Binary Arithmetic Coder. IBM Journal of research and development, Vol. 32, No. 6, pp. 771-726, November 1998.

[PNG] ISO/IEC 15948, Information technology – Computer Graphics and Image Processing – Portable Network Graphics (PNG): Functional specification, 2003.

[Przel05] A. Przelaskowski, Kompresja danych. Wydawnictwo BTC, Warszawa, Polska, 2005.

[Purn12] N. Purnachand, L. N. Alves and A. Navarro, "Improvements to TZ search motion estimation algorithm for multiview video coding," *2012 19th International Conference on Systems, Signals and Image Processing (IWSSIP)*, Vienna, 2012, pp. 388-391.

[Rao06] K. Rao, S. Kwon, A. Tamhankar, Overview of H.264/MPEG-4 Part 10, Journal of Visual Communication and Image Representation, Volume 17, Issue 2, April 2006, pp. 186-216.

[Rice79] R. F. Rice, Some Practical Universal Noiseless Coding Techniques. Jet Propulsion Laboratory, Pasadena, California, JPL Publication 79-22, March 1979.

[Richa03] I. E. G. Richardson, H.264 and MPEG-4 Video Compression. John Wiley & Sons, 2003.

[Richa10] I. E. Richardson, The H.264 Advanced Video Compression Standard. Second edition, John Wiley and Sons, 2010.

[Riss76] J. J. Rissanen, Generalized Kraft Inequality and Arithmetic Coding. IBM Journal of Research and Development, Vol. 20, pp. 198-203, May 1976.

[Riss79] J. J. Rissanen, and G. G. Langdon, Arithmetic Coding. IBM Journal of Research and Development, Vol. 23, No. 2, pp. 149-162, March 1979.

[Said04] A. Said, Introduction to Arithmetic Coding – Theory and Practice. Imaging systems Laboratory, HP Laboratories Palo Alto, April 21, 2004.

[Salom06] D. Salomon, Data Compression. The Complete Reference – 4th Edition. Springer-Verlag, 2006.

[Salom07] D. Salomon, Variable-Length Codes for Data Compression. Springer-Verlag, 2007.

[Salom10] D. Salomon, G. Motta, Handbook of Data Compression. Fifth Edition, Springer-Verlag, 2010.

[Sayo00] K. Sayood, Introduction to Data Compression, 2nd ed. Morgan Kaufmann, 2000.

[Sayo12] K. Sayood, Introduction to Data Compression, 4th ed. Morgan Kaufmann, 2012.

[Shan48] C. E. Shannon, A Mathematical Theory of Communications. Bell System Technical Journal, Vol. 27, pp. 379-423, 623-656, 1948.

[Skarb98] W. Skarbek, Multimedia. Algorytmy i Standardy Kompresji. Akademicka Oficyna Wydawnicza PLJ, Warszawa, 1998.

[SMPTE 240M] SMPTE Standard for Television: 1125-line high definision production systems – signal parameters, ANSI/SMPTE 240M- 1999.

[Stan16] Jakub Stankowski, Damian Karwowski, Krzysztof Klimaszewski, Krzysztof Wegner, Olgierd Stankiewicz, Tomasz Grajek, Analysis of the complexity of the HEVC motion estimation, 23rd International Conference on Systems, Signals and Image Processing IWSSIP 2016, 23-25 May, Bratislava, Slovak Republic.

[Taub02] D. S. Taubman, and M. W. Marcellin, JPEG2000 Image Compression Fundamentals. Standards and Practice, Kluwer Academic Publishers, Boston, 2002.

[Teuh78] J. Teuhola, A Compression Method for Clustered Bit-Vectors. Information Processing Letters, Vol. 7, pp. 308-311, October 1978.

[TIFF] Adobe Systems Incorporated, TIFF, revision 6.0. Final – June 3, 1992.

[VC-1] Society of Motion Picture and Television Engineers, VC-1 Compressed Video Bitstream Format and Decoding Process, SMPTE 421M-2006, 2006.

[Weg18] Krzysztof Wegner, Damian Karwowski, Jakub Stankowski, Tomasz Grajek, Krzysztof Klimaszewski, Olgierd Stankiewicz, "Fast mode selection in the high-efficiency video coding intravideo encoder based on statistics of modes," Journal of Electronic Imaging 27(4), 043051 (23 August 2018). https://doi.org/10.1117/1.JEI.27.4.043051 . Submission: Received: 1 November 2017; Accepted: 24 July 2018.

[Welch84] T. A. Welch, A Technique for High-Performance Data Compression. IEEE Computer, pp. 8-19, June 1984.

[Wieg03] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, Overview of the H.264/AVC Video Coding Standard. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 7, pp. 560-576, July 2003.

[Will95] F. M. J. Willems, Y. M. Shtarkov, and Tj. J. Tjalkens, The Context-Tree Weighting Method: Basic Properties. IEEE Transactions on Information Theory, Vol. 41, No. 3, pp. 653-664, May 1995.

[Will98a] F. M. J. Willems, The Context-Tree Weighting Method: Extensions. IEEE Transactions on Information Theory, Vol. 44, No. 2, pp. 792-797, March 1998.

[Witt87] I. H. Witten, J. G. Cleary, and R. Neal, Arithmetic Coding for Data Compression. Commun ACM., no. 6, pp. 520-540, June 1987.

[Wysz82] G. Wyszecki, and W. S. Styles, Color Science: Concepts and Methods, Quantitative Data and Formulae. Second Edition, Wiley 1982.

[Zhu00] Shan Zhu and Kai-Kuang Ma, "A new diamond search algorithm for fast block-matching motion estimation," in *IEEE Transactions on Image Processing*, vol. 9, no. 2, pp. 287-290, Feb 2000. doi: 10.1109/83.821744.

[Ziv77] J. Ziv, and A. Lempel, A Universal Algorithm for Data Compression. IEEE Transactions on Information Theory, Vol. IT-23, No. 3, pp. 337-343, May 1977.

[Ziv78] J. Ziv, and A. Lempel, A Universal Algorithm for Data Compression. IEEE Transactions on Information Theory, Vol. IT-24, No. 5, pp. 530-536, September 1978.