Adam Łuczak
Sławomir Maćkowiak
Instytut Elektroniki i Telekomunikacji
Politechnika Poznańska
Ul. Piotrowo 3A
60-965 Poznań
tel. 879-10-16 w 171, fax 878-25-72
e-mail: [aluczak,smack]@et.put.poznan.pl

UNIWERSALNE OPROGRAMOWANIE DO REALIZACJI KODERÓW WIZYJNYCH

Streszczenie: **Artykuł prezentuje uniwersalne oprogramowanie do realizacji koderów wizyjnych np. H.263, MPEG-2. Zaprojektowane i stworzone w celu wsparcia istniejących badań i opracowywania nowych algorytmów kompresji sygnałów wizyjnych. Zaprezentowano przykład implementacji fragmentu kodera za pomocą zaproponowanego oprogramowania. Wskazano na cechy charakteryzujące uniwersalność i dużą elastyczność wykorzystania obiektowych metod konstrukcji oprogramowania w badaniach jako efektywnego narzędzia pracy.**

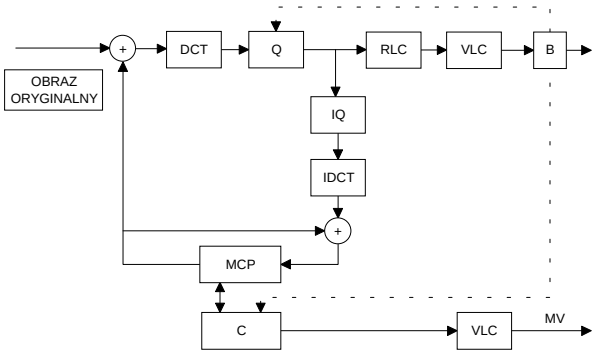
1. WPROWADZENIE

W Zakładzie Komunikacji Multimedialnej i Radiokomunikacji Politechniki Poznańskiej przeprowadzane są badania w zakresie projektowania nowych algorytmów kompresji sygnałów wizyjnych oraz modyfikacji technik ujętych już w standardy. Zestaw stacji roboczych SUN SparcStation, Ultra, cyfrowy sekwencer wizyjny ProntoVideo wraz z sprzętem audiowizualnym (monitor, magnetowid) umożliwia przeprowadzanie badań na sekwencjach wizyjnych o rozdzielczości telewizyjnej zgodnej ze standardem ITU-R BT.601. Aby możliwe było pełne wykorzystanie potencjału w postaci specjalizowanego sprzętu wymagane jest odpowiednie oprogramowanie. Potrzeba takiego oprogramowania wynika przede wszystkim z konieczności eksperymentalnych weryfikacji koncepcji, teorii i założeń.

Istnieje kilka dostępnych, przykładowych implementacji koderów wizyjnych [7],[11] opartych o standardy ISO/IEC 13818-2 (MPEG-2) [4] i H.263 [5]. Niestety realizacje te obarczone są kilkoma poważnymi wadami. Są nimi: trudność modyfikacji takiej implementacji przy wprowadzaniu nowego proponowanego algorytmu, wynikająca z braku odpowiednich komentarzy przy procedurach, niespójność procedur autorstwa wielu osób, oraz sam proceduralny sposób pisania programów. Wiele przeprowadzanych badań właśnie na takim oprogramowaniu przyczyniło się do napisania uniwersalnego oprogramowania, które z założenia będzie pozbawione powyższych wad.

2. STANDARDOWE KODERY WIZYJNE

Poniżej zostanie skrótowo przedstawiona powszechnie używana metoda kompresji przyjęta w większości różnych standardów np. H.261, H.263, MPEG-1, MPEG-2, MPEG-4 (core) przeznaczonych dla różnych przepływności, zarówno małych np. 16 kb/s, jak i wielkich 50 Mb/s. Ma ona duże znaczenie z punktu widzenia założeń projektu ponieważ porównując struktury tych koderów wizyjnych można wyciągnąć bardzo ważny wniosek: algorytm kodowania oparty jest na wspólnym schemacie kodera hybrydowego (rys. 1). Różni się on jedynie w szczegółach uzależnionych od realizowanego zadania. Tym samym oprogramowanie może pełnić uniwersalną funkcję będąc podstawą dla wielu istniejących jak i nowych algorytmów koderów wizyjnych.



Rys. 1 Schemat kodera hybrydowego

Termin kodowanie hybrydowe podkreśla łączne stosowanie wielu metod, które są modyfikowane i przełączane w zależności od treści obrazu. W opisywanej metodzie trzy składowe (luminancja Y oraz chrominancje U i V) są kodowane niezależnie, jednak z wykorzystaniem tej samej informacji o ruchu obiektów uzyskiwanej wyłącznie ze składowej luminancji.

Obrazy są dzielone na nie nakładające się na siebie bloki o rozmiarze 8x8 punktów jednej składowej. Cztery sąsiadujące bloki luminancji tworzą makroblok.

Makrobloki kodowane są na dwa zasadnicze sposoby: wewnątrzobrazowy i międzyobrazowy.

Kodowanie wewnątrzobrazowe odbywa się niezależnie dla każdego obrazu, podczas, gdy kodowanie międzyobrazowe wykorzystuje podobieństwa między kolejnymi obrazami.

Kodowanie wewnątrzobrazowe stosuje się do wszystkich makrobloków pierwszego obrazu sekwencji i pierwszego obrazu w grupie obrazów GOP [2], który jest punktem od którego można odtwarzać sekwencję bez potrzeby jej dekodowania od początku. Dla pozostałych obrazów koder szacuje efektywność obu rodzajów kodowania i wybiera korzystniejszy.

Kodowanie międzyobrazowe jest korzystniejsze zwłaszcza dla statycznych partii obrazu a także dla obiektów o umiarkowanym ruchu. W kodowaniu wewnątrzobrazowym stosuje się kodowanie transformatowe oparte na podziale obrazu na bloki o rozmiarze 8x8 punktów, dla których wyznacza się transformatę kosinusową. Współczynniki transformaty są następnie kwantowane i poddawane kodowaniu ciągów oraz entropowemu.

Kodowanie międzyobrazowe korzysta w zasadzie z tych samych technik co kodowanie wewnątrzobrazowe. Zasadnicza różnica polega na tym, że przetwarzaniu podlega nie bezpośrednio obraz, lecz błąd predykcji obrazu. Zazwyczaj wykorzystuje się jednak bardziej zaawansowaną metodę tak zwanej predykcji z kompensacją ruchu (*motion compensated prediction*). Jej pierwszym, ale za to bardzo pracochłonnym krokiem jest estymacja ruchu. Dla kwadratów 16x16 próbek luminancji należących do poszczególnych makrobloków wyszukuje się w poprzednim obrazie najbardziej podobne do nich kwadraty złożone z 16x16 punktów luminancji. Różnice położeń między odpowiadającymi sobie kwadratami o rozmiarze 16x16 punktów wyznaczają wektory ruchu (MV – *motion vector*).

Z wykorzystaniem kompensacji ruchu lub bez niej, wyznaczany jest przewidywany obraz, który porównuje się z rzeczywistym obrazem na wejściu kodera. Różnica między tymi dwoma obrazami określa błąd predykcji podlegający kodowaniu podobnemu do kodowania wewnątrzobrazowego. Składający się z bloków o rozmiarze 8x8 punktów sygnał błędu predykcji jest poddawany dwuwymiarowej transformacji kosinusowej. Współczynniki uzyskanej transformaty są kwantowane, a następnie wewnątrz każdego z bloków szeregowane zgodnie z uporządkowaniem zygzakowatym. Ciągi współczynników transformaty poddaje się kodowaniu ciągów RLC, a następnie kodowaniu o zmiennej długości słowa VLC, np. kodowaniu Huffmana [3]. Tak przygotowana informacja dociera przez kanał do dekodera.

3. KONCEPCJA UNIWERSALNEGO OPROGRAMOWANIA

Głównym celem leżącym u podłoża powstania oprogramowania jest badanie nowych, proponowanych algorytmów. Każdy nowy algorytm nie jest na samym początku idealny w swojej konstrukcji, dlatego też potrzebna jest możliwość szybkiej oraz łatwej zamiany poszczególnych operacji innymi. Wynika stąd pierwsza cecha jaką charakteryzuje się implementacja. Jest nią elastyczność. Kolejną zaletą jest przejrzystość stworzonej realizacji. Oprogramowanie mimo, że jest autorstwa kilku osób jest bardzo przejrzyste i zrozumiałe dla osób analizujących źródłowy tekst. Kod źródłowy jest bardzo zbliżony do konstrukcji algorytmu, przez co możliwa jest szybsza jego analiza. Kolejną ważną cecha jest uniwersalność oprogramowania. Przy pomocy tych samych obiektów i klas możemy realizować wiele różnych rodzajów koderów wizyjnych, oraz w prosty sposób dokonywać mutacji z jednej postaci w drugą.

Istotą oprogramowania jest poprawność jego działania. Realizując konstrukcję danego algorytmu musimy być pewni, że wykorzystując podstawowe bloki oprogramowania wszystkie operacje będą wykonywane poprawnie. Minimalizujemy tym samym czas potrzebny na usunięcie błędów wynikających z konstrukcji algorytmu.

Zrezygnowano w tej chwili z optymalizacji prędkości działania algorytmów, ponieważ wymaga to dużych mocy obliczeniowych oraz optymalizacji operacji ze względu na prędkość co wiąże się z konstruowaniem oprogramowania dedykowanego konkretnej platformie komputerowej (najczęściej wymagane jest wykorzystywanie rozkazów jednostki procesorowej danego systemu, lub budowanie systemu równoległego realizującego równocześnie wiele wątków programu) oraz konkretnemu systemowi operacyjnemu. Ograniczałoby to uniwersalność oprogramowania co jest jednym z punktów założenia.

4. ZAŁOŻENIA PROJEKTU

Głównym celem naszej pracy było stworzenie bardzo efektywnego narzędzia badawczego koderów wizyjnych. Założono, że implementacja zostanie zrealizowana w postaci obiektowej. Zrezygnowano z proceduralnego sposobu pisania programu chcąc uzyskać większą elastyczność oprogramowania, przejrzystość źródła, efektywność oraz możliwość szybkiej modyfikacji różnych parametrów badanego kodera wizyjnego. Założono, że klasy tworzone przez różnych autorów będą mogły być wykorzystywane przez inne osoby. Niweluje to pisanie powtarzających się procedur i fragmentów kodu. Deklaracje metod i klas powinny być zrozumiałe i proste w użyciu, a także funkcjonalne co w pełni umożliwi wykorzystanie danej klasy. Założono także, że na każdej warstwie oprogramowania będą generowane komunikaty w celu uzyskania kontroli działania oprogramowania tak aby w szybki sposób uzyskać informację o miejscu wystąpienia błędu i przyczynie jego powstania.

5. JĘZYK PROGRAMOWANIA

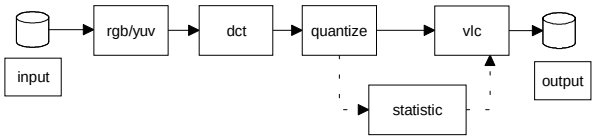
Do realizacji projektu skorzystano z kompilatora GNU CC szeroko rozpowszechnionego w środowisku UNIX. Wybór takiego kompilatora umożliwił stworzenie oprogramowania, które może być

przenoszone między różnymi środowiskami i platformami sprzętowymi, dzięki czemu uzyskuje ono jeszcze większą uniwersalność. Mamy możliwość kompilacji oprogramowania zarówno na komputerach typu SUN jak i PC, w środowisku Solaris lub Windows98/NT. Umożliwiając w ten sposób kompilację źródeł także za pomocą komercyjnych kompilatorów np. Microsoft Visual C++. Jest to duża zaleta z punktu widzenia dydaktycznego, polegająca na udostępnieniu studentom w Laboratorium Cyfrowego Przetwarzania Obrazów oprogramowania bez względu na rodzaj używanego przez nich sprzętu komputerowego.

6. PRZYKŁAD REALIZACJI I MODYFIKACJI ALGORYTMU KODOWANIA OBRAZU

Poniżej przedstawiono przykłady kodowania obrazu w celu zilustrowania uniwersalności modułów oraz łatwości ich używania.

Przykład pierwszy pokazany na rys.2 koduje obraz z użyciem dwuwymiarowej dyskretnej transformaty kosinusowej. Jej wynik jest kwantowany a na podstawie uzyskanych współczynników wyznaczana jest statystyka, która następnie służy skonstruowaniu kodów Huffmana (VLC). Skwantowane próbki DCT poddawane są zamianie na odpowiadające im kody zmiennej długości VLC.



Rys.2 Przykładowy schemat algorytmu kodowania obrazu

Przez to, że realizacja oprogramowania jest obiektowa, schemat blokowy kodera w dużej mierze implikuje nam deklaracje obiektów wykorzystywanych w programie (rys.3a). Natomiast graf przepływu danych wynikający ze schematu kodera w prosty sposób przekłada się na główną część programu. Brak w niej operacji nie związanych z algorytmem kodowania (np. alokacja pamięci, ustawianie zmiennych, sprawdzanie zakresów) wpływa na jej przejrzystość. Umożliwia to szybkie znalezienie błędów algorytmu i jego modyfikacje.

a)

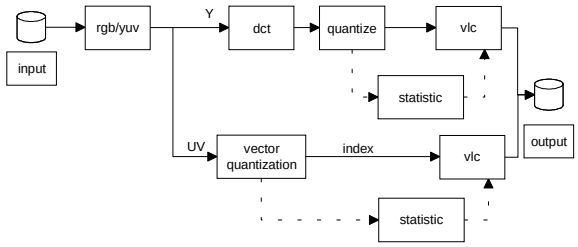
```
ImgRGB rgb(512,512);
ImgYUV yuv(512,512,CHROMA_FORMAT_420)
fstream input(„in.ppm”,ios::in);
fstream output(„out.bin”,ios:out);
Cstat statistic;
CVLC vlc;
CBitstream bitstream;
```

b)

```
main
{
  bitstream.AssignOutput(output);
  vlc.AssignOutput(bitstream);
  rgb.GetPPM(fstream);
  yuv<<rgb;
  yuv.Y-=128;
  yuv.DCT();
  yuv.Quantize(16);
  statistic(1,yuv.Y);
  vlc.Load(statistic);
  // pętla po wszystkich punktach składowej Y
  ...
    vlc.Encode(yuv.Y.Pel[y][x]);
  ...
  statistic(1,yuv.U);
  vlc.Load(statistic);
  // pętla po wszystkich punktach składowej U
  ...
    vlc.Encode(yuv.U.Pel[y][x]);
  ...
  statistic(1,yuv.V);
  vlc.Load(statistic);
  // pętla po wszystkich punktach składowej V
  ...
    vlc.Encode(yuv.V.Pel[y][x]);
  ...
};
```

Rys.3 Realizacja programowa schematu kodera z rys.1

Obraz poddawany przetwarzaniu staje się obiektem. Zaletą takiego podejścia jest możliwość wykonania operacji na jednej lub wszystkich składowych bez konieczności znajomości rozmiaru obrazu, formatu chrominancji czy zakresu dynamicznego próbek.



Rys.4 Zmodyfikowany schemat algorytmu kodowania obrazu

Przykładem prostej modyfikacji algorytmu z rys. 2 jest zmiana sposobu kodowania składowych obrazu U i V (rys. 4).

a)

```
ImgRGB rgb(512,512);
ImgYUV yuv(512,512,CHROMA_FORMAT_420)
fstream input(„in.ppm”,ios::in);
fstream output(„out.bin”,ios:out);
Cstat statistic;
CVLC vlc;
CBitstream bitstream;
CVQ vq;
Component index;
```

b)

```
main
{
  bitstream.AssignOutput(output);
  vlc.AssignOutput(bitstream);
  rgb.GetPPM(fstream);
  yuv<<rgb;
  yuv.Y-=128;
  yuv.Y.DCT();
  yuv.Y.Quantize(16);
  statistic(1,yuv.Y);
  vlc.Load(statistic);
  // pętla po wszystkich punktach składowej Y
  ...
    vlc.Encode(yuv.Y.Pel[y][x]);
  ...
  statistic(2,yuv.U,yuv.V);
  vq.SetRange(16);
  vq.CreateCodeBook(statistic);
  index=vq.Quantize(2,yuv.U,yuv.V);
  vlc.Load(statistic);
  // pętla po wszystkich punktach składowej index
  ...
    vlc.Encode(index.Pel[y][x]);
  ...
};
```

Rys.5 Realizacja programowa schematu kodera z rys.4

Składowa luminancji Y kodowana jest w taki sam sposób jak poprzednio natomiast chrominancje

najpierw kwantujemy wektorowo a na podstawie uzyskanych indeksów książki kodowej są tworzone kody entropowe.

Zmodyfikowany fragment (rys.5 wyróżniony tekst) różni się od oryginału tylko czterema liniami mimo znacznej modyfikacji algorytmu kodowania.

7. WERYFIKACJA

Ważnym problemem podczas projektowania oprogramowania jest jego przetestowanie pod kątem prawidłowego działania. Zanim nowa klasa zostanie umieszczona w ogólnodostępnej bibliotece jest najpierw sprawdzana z punktu widzenia poprawności implementacji, kompatybilności z innymi klasami oraz funkcjonalności interface'u użytkownika.

W zależności od funkcji realizowanych przez daną klasę zmienia się sposób testowania i tak na przykład: transformacje bezstratne jak DCT czy DWT są testowane poprzez dokonanie przekształcenia prostego i odwrotnego. Wiele testów niestety musi być przeprowadzonych manualnie. Wspomagane jest to poprzez bardzo rozbudowany system kontroli realizowanych operacji na wielu warstwach oprogramowania. Dokonuje się to poprzez sprawdzanie dopuszczalnych zakresów parametrów oraz poprawności logicznej zmiennych. Na przykład obiekt odpowiedzialny za generowanie kodów VLC sprawdza ich jednoznaczną dekodowalność, chroniąc się przed niepoprawnością logiczną wprowadzanych danych. Jego elastyczność wynika z możliwości wprowadzania kodów w wielu formatach i na różne sposoby.

Istnieją dwa mechanizmy wizualnej kontroli prawidłowości działania programu. Pierwszy z nich polega na generowaniu w sytuacjach krytycznych odpowiednich komunikatów. Realizuje to obiekt *message*, który na podstawie wiadomości przekazywanych przez inne obiekty generuje odpowiednie komunikaty. Istnieją trzy rodzaje wiadomości:
informacja – czyli powiadomienie o poprawnie wykonanej operacji np.:

```
info>[Matrix::Load] Load successfull.(Horizontal)
info>[Matrix::Load] Load successfull.(Vertical)
info>[Filter::Load] Load successfull.(LowPassFilter)
```

ostrzeżenie – informacja o wyniku działaniu metody, nie mieszczącym się w żądanym zakresie ale nie wpływającym na poprawność działania algorytmu np.:

```
warning>[VQ::CreateCodeBook] Real range 16.(VectorQuantizer)
warning>[Component::Operator +] Different size of Components.
```

błąd – powiadomienie występujące w przypadku niemożliwości wykonania operacji lub w przypadku przekroczenia dopuszczalnych wartości parametrów mogących powodować błędne działanie algorytmu np.:

```
error>[Matrix::Load] Section not found! (Horizontal)
error>[Matrix::Load] Section not found! (Vertical)
error>[Filter::Load] Syntax error! (LowPassFilter)
```

Kolejnym z zaimplementowanych sposobów weryfikacji jest generowanie przez obiekt *bitstream* pliku zewnętrznego zawierającego zapis wszystkich wypuszczanych bitów w formacie tekstowym wraz z odpowiednim komentarzem. Zapis ten odzwierciedla bity tworzące strumień bitowy, umożliwiając w prosty sposób sprawdzenie poprawności składni strumienia danych z odpowiednią normą, standardem lub założonym projektem.

8. WNIOSKI

Wykonane oprogramowanie dotychczas umożliwiło zrealizowanie i udostępnienie do badań implementacji koderów H.263, MPEG-2 Main Profile / Main Level oraz jest bardzo pomocnym narzędziem podczas prac związanych z projektowaniem kodera skalowalnego. Biblioteki klas są nadal rozwijane i wzbogacane o nowe techniki przetwarzania obrazów mając na uwadze wprowadzanie i rozwijanie nowych standardów przez co zwiększa się ich funkcjonalność oraz szeroki zakres zastosowań.

Praca została przygotowana w ramach projektu KBN nr. 8T11D 007 11.

9. SPIS LITERATURY

[1] F. Gadegast, The MPEG – Faq, version 4.0, Phade Software,June 07 1995
[2] B. G. Haskell, A. Puri, A. N. Netravali, *Digital Video: an Introduction to MPEG-2*, Chapman & Hall 1997
[3] G. Held, T. R. Marshall, Data and Image Compression – fourth edition, John Wiley & Sons Ltd 1996
[4] International Telecommunication Union, Transmission of non-telephone signals - Information Technology – Generic Coding of Moving Pictures and Associated Audio Information: Video ITU-T Recommendation H.262, ITU-T 7/95
[5] Draft Recommendation H.263, 1995
[6] J. Levine, Programowanie plików graficznych w C/C++, Wydawnictwo Translator s.c, 1994
[7] K. O. Lillevold, TMN5, Telenor Research, 1995
[8] A. Łuczak, Kodek H.263 sygnałów wizyjnych dla kanałów o bardzo małej przepływności, praca magisterska 1997 Politechnika Poznańska
[9] S. Maćkowiak, Kodek telewizji cyfrowej standardu MPEG-2, praca magisterska 1997 Politechnika Poznańska
[10]A. Marciniak, Borland C++, 1993
[11]MPEG Software Simulation Group, MPEG-2 Encoder Decoder version 1.2, July 19, 1996, http://www.mpeg.org/MSSG/
[12]TMN Encoder, http://www.telenor.no
[13]P. S. Wang, An Introduction to ANSI C on UNIX, Wadsworth Publishing Company, 1992
[14]R. S. Wiener, L. J. Pinson, An Introduction to Object-Oriented Programming and C++, Addison-Wesley Publishing Company,Inc, 1998-11-26