SKRYPT DO LABORATORIUM

WYMIANA I SKŁADOWANIE DANYCH MULTIMEDIALNYCH

ĆWICZENIE 5: Wymiana danych w formacie JPEG2000

autor:

dr inż. Jacek Rumiński

Gdańsk, 2010

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt *„Przygotowanie i realizacja kierunku inżynieria biomedyczna – studia międzywydziałowe”*
współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

1. Zagadnienia wstępne

1.1.Wymagania w odniesieniu do studenta

Osiągnięcie celów ćwiczenia wymaga właściwego przygotowania się studenta do zajęć. Będzie to możliwe poprzez:

- powtórzenie wiedzy nabytej w czasie wykładów,
- zapoznanie się z instrukcją do ćwiczenia ilustrującą podstawowe zagadnienia z zakresu tematyki ćwiczenia, a szczególnie przestrzeni kolorów, splotu sygnałów, filtracji sygnałów z zastosowaniem splotu, transformacji falkowej.

Ponadto wymagana jest od studenta:

- podstawowa umiejętność programowania,
- umiejętność twórczego myślenia.

1.2.Wymagania w odniesieniu do stanowiska laboratoryjnego

Stanowisko laboratoryjne powinno być wyposażone w komputer klasy PC z działającym oprogramowaniem przygotowanym specjalnie dla potrzeb realizacji ćwiczenia. Na komputerze dostępne powinna być zainstalowana instrukcja do ćwiczenia w wersji elektronicznej oraz pliki testowe.

1.3.Cele ćwiczenia

Celem ćwiczenia jest praktyczna demonstracja wiedzy teoretycznej, przedstawionej w ramach wykładów. Realizując ćwiczenie laboratoryjne studenci zdobędą szereg nowych umiejętności w zakresie wymiany danych obrazowych, w tym:

- transformacji danych do systemu kolorów YUV/YCrCb z kompresją,
- praktycznej realizacji operacji splotu,
- praktycznej realizacji operacji kwantyzacji danych w kompresji,
- praktycznej realizacji wielopoziomowej transformacji falkowej z wykorzystaniem filtracji,
- wykorzystania własności progresywnej jakości obrazów,
- wykorzystania własności progresywnej skali obrazów,
- obiektywnej ocenie stopnia i jakości kompresji.

1.4.Metody dydaktyczne

Student w realizuje kolejne zadania opisane w instrukcji. Układ zadań jest tak dobrany, aby zapoznać ćwiczącego z podstawowymi (wybranymi), kolejnymi operacjami wykorzystywanymi w dekompozycji/kompresji obrazów. Ćwiczący ma do dyspozycji kody źródłowe oprogramowania oraz przykładowe obrazy testowe. Wszystkie utworzone dokumenty i zmodyfikowane kody źródłowe programów wpisuje w dokumencie elektronicznym, który jednocześnie staje się sprawozdaniem z ćwiczenia laboratoryjnego.

1.5.Zasady oceniania

Ocenie podlegać będzie realizacja poszczególnych zadań w ramach danego ćwiczenia laboratoryjnego. Dodatkowo w czasie realizacji ćwiczenia przydzielone zostaną studentowi dodatkowe zadania. Student musi zrealizować ćwiczenie laboratoryjne. Dodatkowym warunkiem zaliczenia jest uzyskanie minimum 16 punktów na podstawie oceny sprawozdań/plików źródłowych.

Wykaz literatury podstawowej do ćwiczenia:

Wykaz literatury podstawowej:

1.	Skrypt z materiałami do przedmiotu „Wymiana i składowanie danych multimedialnych”
2.	Materiały do przedmiotu opracowane w formie edukacji na odległość, dostęp: http://uno.biomed.gda.pl
3.	David Taubman (Editor), Michael Marcellin (Editor), JPEG2000: Image Compression Fundamentals, Standards and Practice, Springer, 2001.

Wykaz literatury uzupełniającej:

2. Przebieg ćwiczenia

L.p.	Zadanie
1.	Zapoznanie się z instrukcją laboratoryjną
2.	Transformacja kolorów z RGB na YUV/YCrCb.
3.	Kompresja luminancji/chrominancji
4.	Realizacja splotu sygnałów
5.	Wykonywanie wielopoziomowej transformacji Falkowej z wykorzystaniem filtracji
6.	Wykonywanie kwantyzacji, w tym kwantyzacji płaszczyzn bitowych
7.	Dekompresja danych
8.	Demonstracja wykorzystania własności progresji skali i jakości obrazów

UWAGI!

1. PRZED przystąpieniem do ćwiczenia należy w katalogu d:\dydaktyka\studenci utworzyć własny katalog (imie_nazwisko), a następnie przekopiować tam zawartość katalogu d:\dydaktyka\wisd\cw5. Proszę pracować wyłącznie na wykonanej kopii plików.

3. Wprowadzenie do ćwiczenia

W czasie realizacji ćwiczenia realizowane będą zadania zgodnie z ich wcześniejszym opisem. Każde zadanie (z wyjątkiem zadania realizowanego przed przystąpieniem do ćwiczenia laboratoryjnego) opisano poniżej.

Ad 1. Zapoznanie się z instrukcją laboratoryjną (5 min.)

Po wprowadzeniu kierownika ćwiczenia uczestnicy zapoznają się ze stanowiskiem komputerowym, oprogramowaniem i dokumentami związanymi z ćwiczeniem.
Następnie utworzyć we wskazanym przez kierownika ćwiczenia katalogu własny podkatalog o nazwie zawierającej własne nazwisko. Do katalogu tego przegrać zawartość podkatalogu „DANE” (znajdującego się w folderze zawierającym instrukcję do ćwiczenia). W utworzonym podkatalogu należy przechowywać wszystkie wytworzone w czasie trwania ćwiczenia dokumenty i programy.

Ad 2. Transformacja kolorów z RGB na YUV/YCrCb (15 min.)

Jednym z podstawowym kroków kompresji obrazów jest przetwarzanie wstępne, w którym dokonuje się transformacji przestrzeni kolorów obrazu. Wykorzystuje się bowiem własność systemu wzrokowego człowieka, zgodnie z którą człowiek jest bardziej wrażliwy na zmiany luminancji niż chrominancji. Standardowo każdy obraz w ogólnym formacie komputerowych (np. JPEG, PNG) jest przechowywany zgodnie z modelem RGB. Dlatego konieczne jest przejście na przestrzeń kolorów z oddzieloną składową luminancji, np. YUV.
Poniżej przedstawiono funkcje realizacji konwersji danych pomiędzy systemami RGB, a YUV (ColorSpaceTransform.java).

```
public static int[] rgb2yuv2(int R,int G, int B)   {
    int[] yuv = new int[3];
    yuv[0]=  (int)(R *  0.299000 + G *  0.587000 + B *  0.114000);
    yuv[1] = (int)(R * -0.168736 + G * -0.331264 + B *  0.500000 + 128);
    yuv[2] = (int)(R *  0.500000 + G * -0.418688 + B * -0.081312 + 128);
    return yuv;
  }

  public static int[] yuv2rgb2(int Y, int U, int V)   {
    int[] rgb = new int[3];
    int C,D,E;
    C= Y - 16;
```

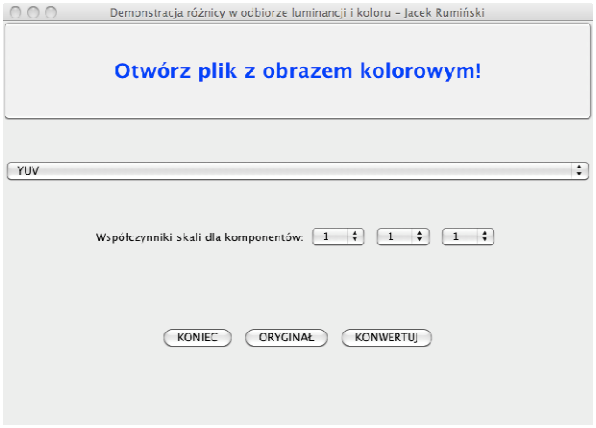
```
    D = U - 128;
    E = V - 128;
    rgb[0]  = scale((int)(Y + 1.4075 * (E)));
    rgb[1] = scale((int)(Y - 0.3455 * (D) - (0.7169 * (E))));
    rgb[2] = scale((int)(Y + 1.7790 * (D)));
    return rgb;
}
```

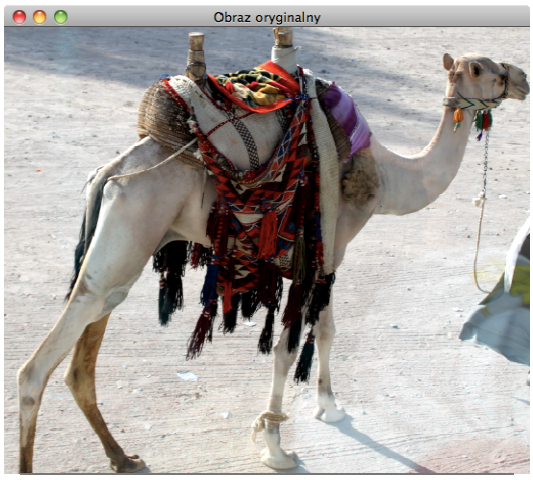
Napisać funkcje transformacji kolorów (w klasie ColorSpaceTransform) pomiędzy systemami kolorów RGB i YCrCb.

Ad 3. Kompresja luminancji/chrominancji (25 min.)

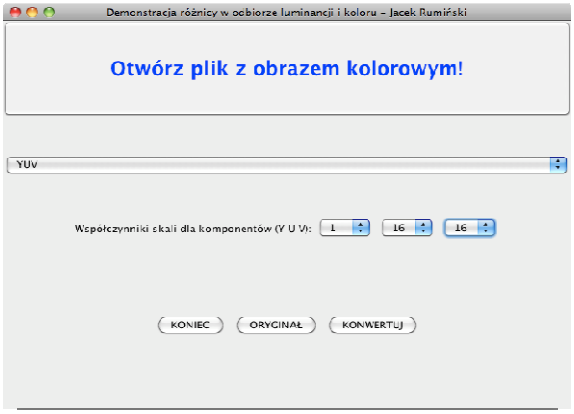
Obraz kolorowy opisany w trójwymiarowym systemie kolorów Y UV podlegać będzie stratnej kompresji poprzez próbkowanie (wybierzemy co n-ty element z macierzy danych). Odtwarzając dane (dekompresja) będziemy powielać tę samą wartość tyle razy, ile wartości opuściliśmy w fazie próbkowania. Przykładowo wybierając współczynnik 16, wybierzemy co 16 wartość WYBRANEJ (Y i/lub U i/lub V) macierzy danych. Odtwarzając obraz o rozdzielczości pierwotnej będziemy powielać 15 razy tę samo, wybraną (w fazie próbkowania) wartość.

Uruchomimy najpierw aplikację luminanceDemo.bat, a następnie otworzymy plik z obrazem kolorowym wielbłąda (wielbłąd.jpg). Po wybraniu przycisku „ORYGINAŁ” pojawi nam się oryginalny obrazek.

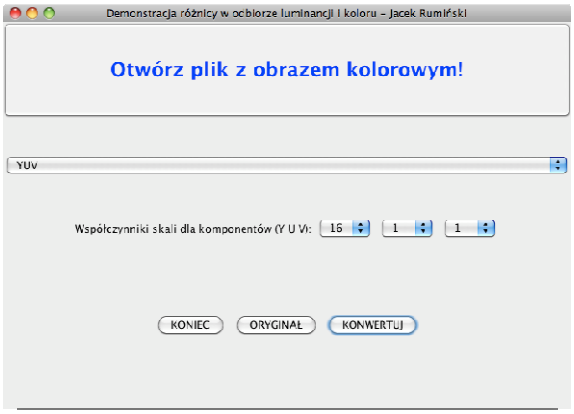


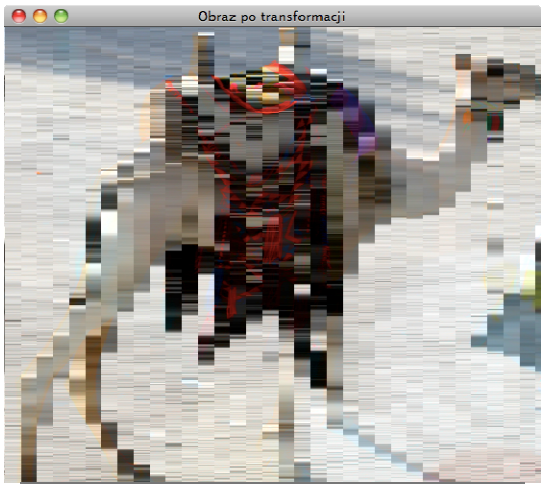


Następnie wybierzemy współczynniki skali dla komponentów U i V (kompresja składowych koloru). Po wybraniu przycisku „KONWERTUJ” pojawi nam się wersja obrazu oryginalnego po kompresji stratnej (co n-ta wartość z macierzy U i V) i dekompresji (rekonstrukcji, powieleniu wartości). Efekt pokazano poniżej.



W kolejnym kroku wybierzemy współczynniki skali >1 jedynie dla komponentu Y (kompresja w torze luminancji). Po wybraniu przycisku KONWERTUJ” pojawi nam się wersja obrazu oryginalnego po kompresji stratnej (co n-ta wartość z macierzy Y) i dekompresji (rekonstrukcji, powieleniu wartości). Efekt pokazano poniżej.





Wykonać eksperymenty wybierając współczynnik skalowania (co która próbkę wybieramy) na 64. Jaki można wysnuć wniosek?

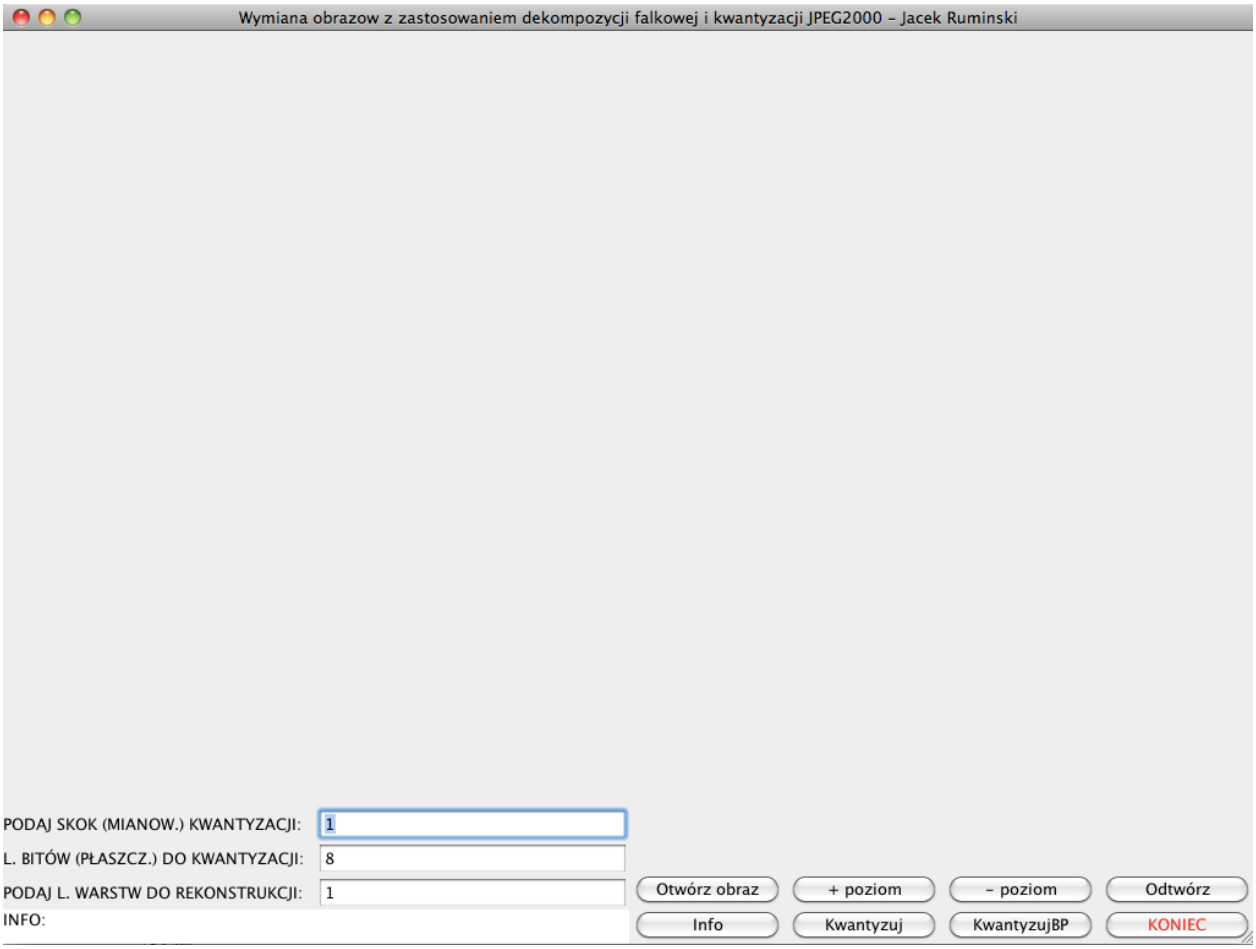
Ad 4. Realizacja splotu sygnałów (15 min.)

Zapoznać się z kodem źródłowym Convolution.java realizującym operację splotu. Dla zadanego przez prowadzącego sygnału cyfrowego sprawdzić efekt splotu z maską filtru zapisaną w programie. Po

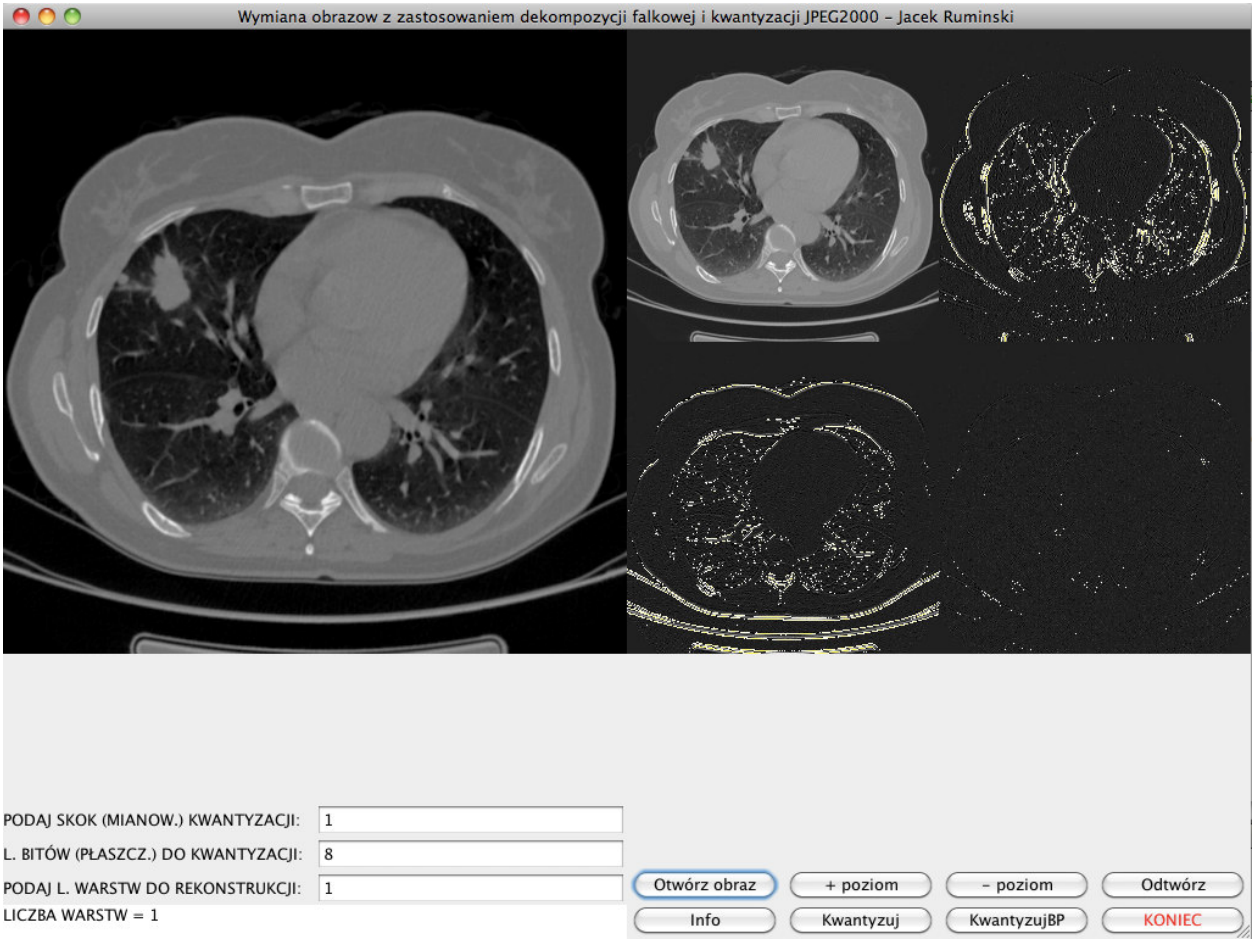
wprowadzeniu zmian (jedynie w metodzie *main()*) skompilować i uruchomić program. Wynik splotu zapisać.

Ad. 5. Wykonywanie wielopoziomowej transformacji Falkowej z wykorzystaniem filtracji (25 min.)

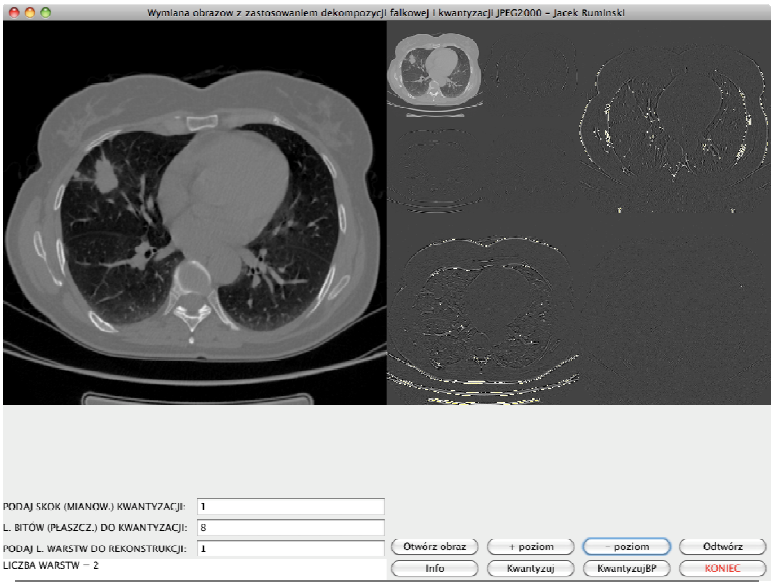
Po zapoznaniu się z kodem źródłowym oprogramowania (*Jpeg2000DEMO*), należy uruchomić program (*jpeg2000DEMO.bat*). Po uruchomieniu się programu pojawi się główne okno:

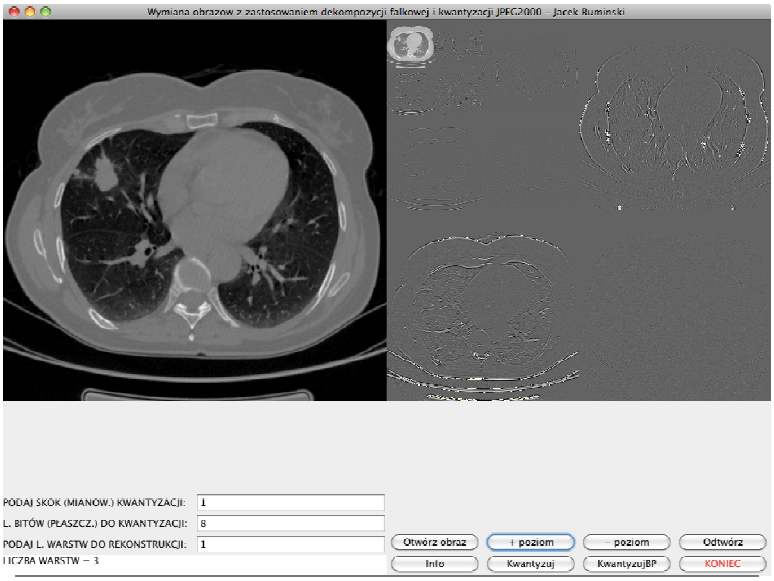


Najpierw należy wczytać obraz z pliku. W tym celu wybieramy przycisk „Otwórz obraz” i wskazujemy przykładowy plik „ct_test.jpg”. Po potwierdzeniu wyboru (w oknie dialogowym) obraz oraz jego jednopoziomowa dekompozycja pojawią się w głównym oknie programu.



Posługując się przyciskami „+ poziom” i „- poziom” można zwiększać liczbę poziomów dekompozycji obrazu (uwaga: mogą pojawić się zmiany odcienia tła, co jest spowodowane dynamicznym skalowaniem zakresu wartości w poszczególnych kanałach dekompozycji tak, aby uczestnik ćwiczeniu mógł zaobserwować wzmocnione wartości w kanałach będących wynikiem filtracji górnoprzepustowej).





Na podstawie kodu źródłowego wyjaśnić jak realizowana jest dekompozycja obrazu w obrębie pojedynczego poziomu.

Ad. 6/7. Wykonywanie kwantyzacji, w tym kwantyzacji płaszczyzn bitowych i dekompresja danych (20 min.)

1. Kwantyzacja klasyczna

Na podstawie kodu źródłowego zapoznać się z zastosowaną metodą kwantyzacji klasycznej. Po otwarciu obrazu (lub wciśnięciu przycisku „+ poziom” uzyskując jednopoziomową dekompozycję obrazu) wykorzystując oprogramowanie wprowadzić wartość mianownika kwantyzacji = 20. Następnie

wcisnąć przycisk „Kwantyzuj”. Zmieniona wartość macierzy danych jest przechowywana w pamięci. W celu dekompresji i uzyskania obrazu należy wcisnąć przycisk „Odtwórz”. W rezultacie utrzymamy obraz przedstawiony poniżej.

Przeprowadzić dwie próby dla różnych wartości mianownika kwantyzacji.

2. Kwantyzacja z wykorzystaniem płaszczyzn bitowych

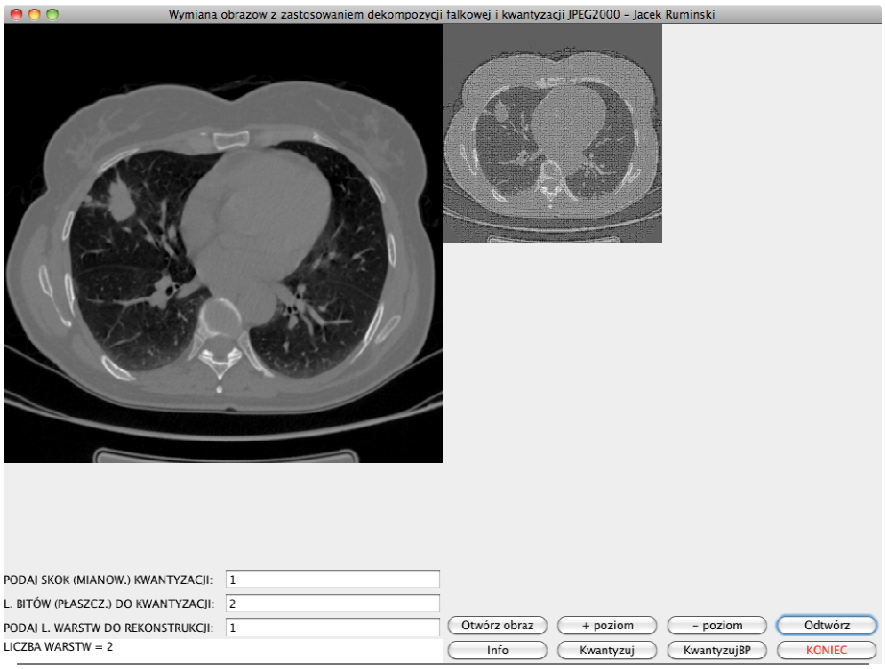
Zapoznać się z realizacją kwantyzacji z wykorzystaniem płaszczyzn bitowych w kodzie źródłowym oprogramowania. Następnie przeprowadzić analogiczną procedurę (ustawić parametr kwantyzacji->KwantyzujBP->Odtwórz) jako we wcześniejszym punkcie zmieniając wartość liczby bitów do kwantyzacji. Przykładowo wybierając wartość 3 pozostawiamy trzy najstarsze bity, a pozostałe są pomijane. W odtwarzaniu zostaną dopisane zera (pięć zer na najmłodszych bitach). Poniżej pokazano przykład wykorzystując 2 najstarsze bity w kwantyzacji.

Należy wykonać dwie próby wykorzystując różną liczbę płaszczyzn bitowych.

Ad. 8. Demonstracja wykorzystania własności progresji skali i jakości obrazów (30 min.)

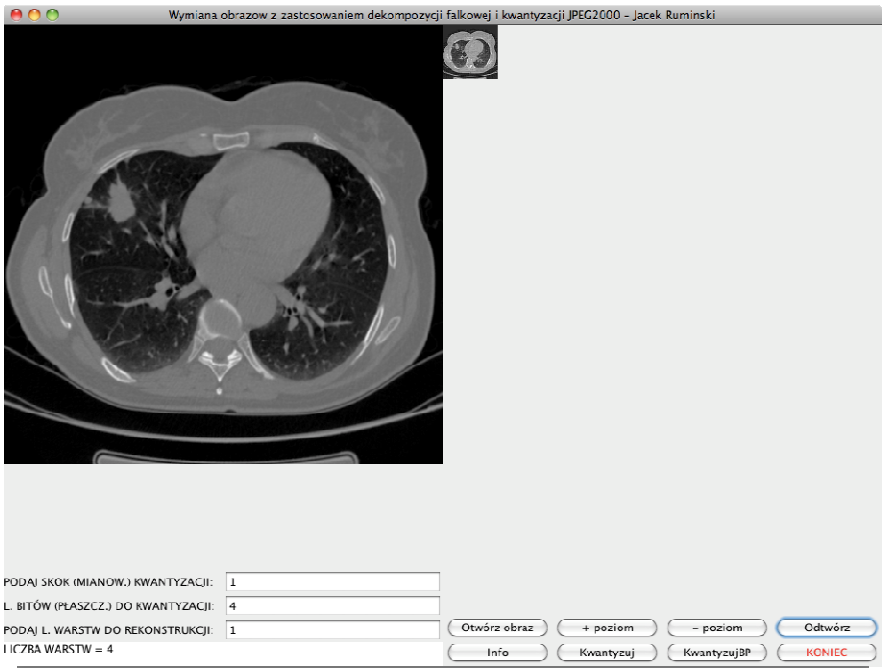
Progresja jakości (dociąganie kolejnych płaszczyzn bitowych) oraz progresja skali (dociąganie kolejnych składowych z dekompozycji obrazu) to niezwykle istotna cecha wykorzystywana w algorytmie JPEG 2000. Jest to szczególnie ważne w aplikacjach medycznych. Szczególnie może być to istotne w analizie wykorzystującej regiony zainteresowania (ang. ROI – region of interest). Przykładowo w obrębie regionu obrazu można inaczej zakodować dane niż w pozostałej części. W ten sposób jeśli pobierany jest obraz ze serwera przez limitowane łącze wówczas tło obrazu może być pobrane z niższą jakością, na rzecz lepszej jakości danych z regionu (np. analizowanej zmiany chorobowej).

W celu obserwacji własności progresji skali i jakości należy po wczytaniu obrazu wygenerować dwupoziomową dekompozycję obrazu (pierwszy poziom automatycznie po otwarciu, drugi po wciśnięciu przycisku „+ poziom”). Następnie należy ustawić liczbę bitów (płaszczyzn bitowych) na 2, po czym wcisnąć przycisk „KonwertujBP”. W celu rekonstrukcji obrazu należy podać liczbę warstw do rekonstrukcji (sterowanie rozdzielczością) i nacisnąć przycisk „Odtwórz”. W efekcie uzyskamy rezultat pokazany poniżej.



Należy zauważyć, że odtworzenie większej rozdzielczości wymagałoby jedynie dociągnięcie trzech pozostałych macierzy po dekompozycji 1 poziomu. Odtworzenie pierwotnej jakości byłoby możliwe poprzez „dociągnięcie” pozostałych płaszczyzn bitowych.

Poniższy rysunek ilustruje efekt odtworzenia obrazu dla 4-poziomowej dekompozycji obrazu.



Wykonać szereg własnych eksperymentów dobierając parametry liczby warstw dekompozycji obrazu oraz liczbę płaszczyzn bitowych.

4 Załączniki

W części tej zawarto kody programów oraz pliki przykładowe i instrukcje konfiguracji oprogramowania.

4.1 Kod źródłowy przykładu kompresji składowych luminancji/chrominancji

// ColorSpaceTransform.java

```
package org.cemte.jacek.mpo;

/**
 *
 * @author jwr
 */
public class ColorSpaceTransform {
   public static int[] rgb2yuv(int r,int g, int b)
   {
      int[] yuv = new int[3];

      yuv[0] = (int)(0.299 * r + 0.587 * g + 0.114 * b)+16;
      yuv[1] = (int)((b - yuv[0]) * 0.492f)+128;
      yuv[2] = (int)((r - yuv[0]) * 0.877f)+128;

      return yuv;
   }

   public static int[] yuv2rgb(int y, int u, int v)
   {
      int[] rgb = new int[3];

      rgb[0] = (int)(1.164*(y - 16) + 1.596*(v - 128));
      rgb[1] = (int)(1.164*(y - 16) - 0.813*(v - 128) - 0.391*(u - 128));
      rgb[2] = (int)(1.164*(y - 16) + 2.018*(u - 128));

      return rgb;
   }

   public static int[] rgb2yuv2(int R,int G, int B)
   {
      int[] yuv = new int[3];

      yuv[0]=  (int)(R *  0.299000 + G *  0.587000 + B *  0.114000);
      yuv[1] = (int)(R * -0.168736 + G * -0.331264 + B *  0.500000 + 128);
      yuv[2] = (int)(R *  0.500000 + G * -0.418688 + B * -0.081312 + 128);


      return yuv;
   }
```

```
   public static int[] yuv2rgb2(int Y, int U, int V)
   {
      int[] rgb = new int[3];
      int C,D,E;
      C= Y - 16;
      D = U - 128;
      E = V - 128;
      rgb[0]  = scale((int)(Y + 1.4075 * (E)));
      rgb[1] = scale((int)(Y - 0.3455 * (D) - (0.7169 * (E))));
      rgb[2] = scale((int)(Y + 1.7790 * (D)));


      return rgb;
   }

   public static int scale(int value){
      if(value<0) return 0;
      else if(value>255) return 255;
      else return value;
   }

}//class
```

// LuminanceColorDemo.java

```
package org.cemte.jacek.mpo;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.*;
/**
*  Ilustracja własności kompresji luminancji/chrominancji
* @author jwr

 */
public class LuminanceColorDemo extends JFrame{
   ImageOps ops=null;
   JFileChooser jfc_chooser=null;
   JButton jb_openFile=null;
   JComboBox jcb_colorSystems=null;
   JComboBox jcb_scaleFactorC1=null;
   JComboBox jcb_scaleFactorC2=null;
   JComboBox jcb_scaleFactorC3=null;

   JButton jb_transform=null;
   JButton jb_exit=null;
   JButton jb_clear=null;

   File pathToImage=null;
   String colorSystems[]={"YUV", "YCrCb"};
   String scaleFactors[]={"1","2","4","8","16","64"};

   Image outImage=null;
```

```
int [] outData=null;

int chosenCS=-1;
int cF1=1,cF2=1,cF3=1;
int chosenFactor[]=new int[3];

public LuminanceColorDemo(String txt){
   super(txt);
   this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
   ops=new ImageOps();
   initGUI();
}

/** budowa układu interfejsu graficznego */
public void initGUI(){
   jfc_chooser = new JFileChooser();
   //JFrame jf = new JFrame("Demonstracja różnicy w odbiorze luminancji i koloru");

   setLayout(new GridLayout(4,1));
   jb_openFile=new JButton("Otwórz plik z obrazem kolorowym!");
   jb_openFile.setFont(new Font("Dialog", Font.BOLD, 24));

   jb_openFile.setForeground(Color.blue);
   jb_openFile.addActionListener(new ActionListener(){
     public void actionPerformed(ActionEvent ae){
        int result = jfc_chooser.showOpenDialog(new JFrame());
        if (result == JFileChooser.APPROVE_OPTION) {
           pathToImage = jfc_chooser.getSelectedFile();
           System.out.println(pathToImage.getAbsolutePath());
        }
     }
   });
   add(jb_openFile);
   jcb_colorSystems=new JComboBox(colorSystems);
   jcb_colorSystems.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
         chosenCS= jcb_colorSystems.getSelectedIndex();
      }
   });
   add(jcb_colorSystems);

   JPanel jp=new JPanel();
   jcb_scaleFactorC1=new JComboBox(scaleFactors);
   jcb_scaleFactorC1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
         cF1= jcb_scaleFactorC1.getSelectedIndex();
      }
   });
   jcb_scaleFactorC2=new JComboBox(scaleFactors);
   jcb_scaleFactorC2.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
         cF2= jcb_scaleFactorC2.getSelectedIndex();
      }
   });
   jcb_scaleFactorC3=new JComboBox(scaleFactors);
```

```
jcb_scaleFactorC3.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      cF3= jcb_scaleFactorC3.getSelectedIndex();
   }
});
jp.add(new JLabel("Współczynniki skali dla komponentów (Y U V):"));
//JPanel pJC=new JPanel
jp.add(jcb_scaleFactorC1);
jp.add(jcb_scaleFactorC2);
jp.add(jcb_scaleFactorC3);

add(jp);

JPanel controlPanel = new JPanel();

jb_exit=new JButton("KONIEC");

jb_exit.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent ae){
      System.exit(0);
   }
});
jb_clear=new JButton("ORYGINAŁ");
jb_clear.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent ae){
     if (pathToImage==null) return;
     int mode=0;
     int scalingFactor=1;

     outImage=ops.readBImage(pathToImage.getAbsolutePath());
     ImageViewPanel ivp=new ImageViewPanel(0,0,ops.width,ops.height,outImage,"Obraz oryginalny");
  }
});

jb_transform=new JButton("KONWERTUJ");
jb_transform.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent ae){
     if (pathToImage==null) return;
     int mode=0;
     int scalingFactor=1;
     if(cF1==1&&cF2==1&&cF3==1){


     }else
     if(cF1>0){
        if(cF2>1||cF3>1){
           mode=7;
        }else{
           mode=4;
        }
        scalingFactor=Integer.parseInt(scaleFactors[cF1]);
     }else{
        if(cF2>1&&cF3>1){
           mode=3;
           scalingFactor=Integer.parseInt(scaleFactors[cF2]);
```

```
            }else{
               if(cF2>1){
                  mode=2;
                  scalingFactor=Integer.parseInt(scaleFactors[cF2]);
               }else{
                  mode=1;
                  scalingFactor=Integer.parseInt(scaleFactors[cF3]);
               }
            }

         }
         System.out.println("TEST: "+mode+", "+scalingFactor);
         outData=convertImageForward(pathToImage);
         outData=convertImageReverse(chromaSubSample(outData,mode,scalingFactor));
         outImage=ImageFactory.create3CImage(ops.width, ops.height, outData);
         ImageViewPanel ivp=new ImageViewPanel(0,0,ops.width,ops.height,outImage,"Obraz po
transformacji");
       }
      });

      controlPanel.add(jb_exit);
      controlPanel.add(jb_clear);
      controlPanel.add(jb_transform);
      add(controlPanel);

   }//initGUI()

  /** zamień odczytane dane obrazu do przestrzeni YUV i dokonaj kompresji, zgodnie z ustawieniami */
   public int[] convertImageForward(File pathToImage){
      int dTmp;
      int[] yuv=new int[3];
      int [] result=null;
      int offset=0;
      if(pathToImage.isFile()){
         BufferedImage bi=ops.readBImage(pathToImage.getAbsolutePath());
         result=new int[ops.width*ops.height];
         for(int i=0;i<ops.height;i++){
            offset=i*ops.width;
            for(int j=0;j<ops.width;j++){
               dTmp=bi.getRGB(j, i);
               yuv=ColorSpaceTransform.rgb2yuv2((dTmp>>16)&255, (dTmp>>8)&255, dTmp&255);
               result[offset+j]=(255<<24)+((yuv[0]&255)<<16)+((yuv[1]&255)<<8)+(yuv[2]&255);
               /*
               if(i==0&&j==0){
                  System.out.println("YUV="+Arrays.toString(yuv));
                  System.out.println("R="+((dTmp>>16)&255)+"G="+ ((dTmp>>8)&255)+"B="+( dTmp&255));
                  System.out.println("Yt="+(yuv[0]<<16)+"Ut="+ (yuv[1]<<8)+"Vt="+(yuv[2]));
                  System.out.println("Result="+result[offset+j]);
               }
                */
               //result[offset+j]=(255<<24)+20;//ColorSpaceTransform.scale(yuv[2]);
            }//for j
         }//for i
      }//if
      //System.out.println(Arrays.toString(result));
```

```
    return result;
  }//convertImageForward(File pathToImage)

 /** odtwórz obraz po kompresji (stratnej) */
 public int[] convertImageReverse(int []data){
    int result[]=null;
    int[] rgb=new int[3];
    int offset=0*ops.width+0;

    result=new int[data.length];
    for(int i=0;i<result.length;i++){
       rgb=ColorSpaceTransform.yuv2rgb2((data[i]>>16)&255, (data[i]>>8)&255, data[i]&255);
       result[i]=(255<<24)+((rgb[0]&255)<<16)+((rgb[1]&255)<<8)+(rgb[2]&255);
       if(i==offset){
            System.out.println("Y="+((data[i]>>16)&255)+"U="+ ((data[i]>>8)&255)+"V="+( data[i]&255));
            System.out.println("RGB="+Arrays.toString(rgb));
            System.out.println("Result="+result[i]);
       }
       //result[i]=255;
    }//for i
    return result;
  }

  public int [] subsample(int [] data, int ch, int factor){
    int [] result=new int[data.length];
    int c1=0,c2=0,c3=0;

    for(int i=0;i<data.length;i++){
       c1=(data[i]>>16)&255;
       c2=(data[i]>>8)&255;
       c3=(data[i])&255;
       switch (ch){
          case 4:
               c1=(int)c1/factor;
               c1=factor*c1;
               break;
          case 2:
               c2=(int)c2/factor;
               c2=factor*c2;
               break;
          case 1:
               c3=(int)c3/factor;
               c3=factor*c3;
               break;
          case 3:
               c2=(int)c2/factor;
               c2=factor*c2;
               c3=(int)c3/factor;
               c3=factor*c3;
               break;
          case 7:
               c1=(int)c1/factor;
               c1=factor*c1;
               c2=(int)c2/factor;
               c2=factor*c2;
```

```
                c3=(int)c3/factor;
                c3=factor*c3;
                break;
            default: break;
        };
        result[i]=(255<<24)+(c1<<16)+(c2<<8)+c3;
    }
    return result;
}//subsample()

/** dokonaj kompresji poprzez redukcję danych – próbkowanie co n-ty element */
public int [] chromaSubSample(int [] data, int ch, int factor){
    int [] result=new int[data.length];
    int c1=0,c2=0,c3=0;

    int prevC1=0,prevC2=0,prevC3=0;
    //factor=16;
    int j=factor;
    for(int i=0;i<data.length;i++){
        c1=(data[i]>>16)&255;
        c2=(data[i]>>8)&255;
        c3=(data[i])&255;
        if((j%factor)!=0){
            /*
            switch(ch){
                case 2: c2=prevC2;
                        c3=prevC3;
                        break;
                case 3: c1=prevC1;
                        break;
                default: break;
            };

             */
            switch(ch){
            case 4:
                    c1=prevC1;
                    break;
            case 2:
                    c2=prevC2;
                    break;
            case 1:
                    c3=prevC3;
                    break;
            case 3:
                    c2=prevC2;
                    c3=prevC3;
                    break;
            case 7:
                    c1=prevC1;
                    c2=prevC2;
                    c3=prevC3;
                    break;
            default: break;
            }
```

```
        }else {
           prevC1=c1;
           prevC2=c2;
           prevC3=c3;
        }
        if(j>1)
           j--;
        else
          j=factor;
        if(i%ops.width==0)
           j=factor;
        result[i]=(255<<24)+(c1<<16)+(c2<<8)+c3;
     }
     return result;
  }


  public static void main(String a[]){
     LuminanceColorDemo lcd=new LuminanceColorDemo("Demonstracja różnicy w odbiorze luminancji i koloru
- Jacek Rumiński");
     int v[]=ColorSpaceTransform.rgb2yuv2(154,231,119);
     System.out.println(Arrays.toString(v));
     int w[]=ColorSpaceTransform.yuv2rgb2(v[0],v[1],v[2]);
     System.out.println(Arrays.toString(w));
     System.out.println("wartosc=" +(1%2));
     lcd.setSize(700,500);
     lcd.setVisible(true);
  }//main()

}//class
```

4.2. Kod źródłowy przykładu realizacji operacji splotu

//Convolution.java

```
import java.util.*;

/**
 *
 * @author  Jacek Ruminski
 */
public class Convolution {

  public static final int NO_EDGE=0;
  public static final int PERIODICAL=1;
  public static final int SYMMETRIC=2;


  /** Creates a new instance of Convolution */
  public Convolution() {
  }

  public double [] convolve1D(double data[],double filter[], int edgeCondition, boolean normalize){
```

```
int edge=filter.length/2;
double result[] = new double[data.length];
int size=data.length;
int filterLength=filter.length;
int reducedSize=size-edge;
int middle=edge+1;
if ((filterLength%2)==0){
   System.out.println("ERROR: Only odd length filters are accepted");
   return null;
}
double tempValue=0;
double normalization=0;
for(int i=0; i<filterLength; i++){
   normalization+=filter[i];
}
if (normalization==0)
   normalization=1;

for(int i=0; i<(size-filterLength+1); i++){
   tempValue=0;
   for(int j=0; j<filterLength; j++){
      tempValue+=data[i+j]*filter[j];
   }
   if (normalize) tempValue/=normalization;
   result[i+edge]=tempValue;
}

double edgeData[] = new double[filterLength];

boolean noOpFlag=false;
//LEFT SIDE
for(int j=0; j<edge; j++){
   switch(edgeCondition){
      case PERIODICAL:
                  for(int i=0; i<(edge-j);i++){
                    edgeData[i]=data[size-edge+i+j];
                  }
                  for(int i=(edge-j),k=0; i<filterLength;i++,k++){
                    edgeData[i]=data[k];
                  }
                  break;
      case SYMMETRIC:
                  for(int i=0; i<(edge-j);i++){
                    edgeData[i]=data[i+1];
                  }
                  for(int i=(edge-j),k=0; i<filterLength;i++,k++){
                    edgeData[i]=data[k];
                  }
                  break;
      default:        noOpFlag=true;
   }//end of switch
   //convolve
   if(!noOpFlag){
      tempValue=0;
      for(int i=0; i<filterLength; i++){
```

```
            tempValue+=edgeData[i]*filter[i];
         }
         if (normalize) tempValue/=normalization;
         result[j]=tempValue;
      } else{
         result[j]=data[j];
      }

   }//end of for j

   //RIGHT SIDE
   for(int j=0; j<edge; j++){
      switch(edgeCondition){
         case PERIODICAL:
                     for(int i=((size-1)-edge-j),l=0; i<size;i++,l++){
                        edgeData[l]=data[i];
                     }

                     for(int i=(edge+1+j),k=0; i<filterLength;i++,k++){
                        edgeData[i]=data[k];
                     }
                     break;
         case SYMMETRIC:
                     for(int i=((size-1)-edge-j),l=0; i<size;i++,l++){
                        edgeData[l]=data[i];
                     }

                     for(int i=(edge+1+j),k=2; i<filterLength;i++,k++){
                        edgeData[i]=data[size-k];
                     }
                     break;
         default:          noOpFlag=true;
      }//end of switch
      //convolve
      if(!noOpFlag){
         tempValue=0;
         for(int i=0; i<filterLength; i++){
            tempValue+=edgeData[i]*filter[i];
         }
         if (normalize) tempValue/=normalization;
         result[size-1-j]=tempValue;
      } else{
         result[size-1-j]=data[size-1-j];
      }

   }//end of for j
   return result;
}//end of double [] convolve1D



public static void main(String a[]){

   Convolution con = new Convolution();
   //double data[] = {10, 20,30,20,10,20,30,20,10,20,30};
```

```
    double data[] = {100, 100,100,100,200,200,200,200};
    double filter[] = {-1.0/8.0,2.0/8.0,6.0/8.0,2.0/8.0,-1.0/8.0};
    System.out.println("Wymiar:" +filter.length);
    //double filter[] = {1.0/5.0,1.0/5.0,1.0/5.0,1.0/5.0,1.0/5.0};

    System.out.println(Arrays.toString(data));
    System.out.println(Arrays.toString(filter));
    System.out.println(Arrays.toString(con.convolve1D(data,filter,Convolution.SYMMETRIC, true)));
  } //main()
}//class
```

4.3. Kod źródłowy przykładu realizacji dekompozycji falkowej obrazu, filtracji i kwantyzacji danych oraz odtwarzania obrazu

// WaveletFilters.java

```
import java.util.*;
/**
 *
 * @author  jwr
 */
public class WaveletFilters {

   public static final int DAUBECHIES_97 = 0;
   public static final int INTEGER_53 = 5;

   public double h0Filter[];
   public double h1Filter[];
   public double g0Filter[];
   public double g1Filter[];
   double indexes_h0 [];
   double indexes_h1 [];
   double indexes_g0 [];
   double indexes_g1 [];

   /*
    * Filtr Daubechies 9,7
    * h0 - low Pass
    */
   public double h0Daubechies9[]={0.026748757410, -0.016864118442, -0.078223266528, 0.266864118441,
0.602949018236, 0.266864118441,-0.078223266528, -0.016864118442, 0.026748757410};

   /*
    * Filtr Daubechies 9,7
    * h1 - high Pass
    */
   public double h1Daubechies7[]={0.091271763114, -0.057543526228, -0.591271763114, 1.115087052456, -
0.591271763114, -0.057543526228,0.091271763114};

   public double h0Int5[]={-1.0/8.0,2.0/8.0,6.0/8.0,2.0/8.0,-1.0/8.0};
   public double h1Int3[]={-1.0/2.0,2.0/2.0,-1.0/2.0};
```

```
   public int currentFilter=-1;
   public double alpha=1;
   private boolean isAlpha=false;

   /** Creates a new instance of WaveletFilters */
   public WaveletFilters() {
   }

/** read analysis filter values */
   public double [] getH0(int filterType){

      switch(filterType){
         case DAUBECHIES_97:
         default:
                  currentFilter=DAUBECHIES_97;
                  h0Filter=h0Daubechies9;
                  break;
         case INTEGER_53:
                  currentFilter=INTEGER_53;
                  h0Filter=h0Int5;
                  //h1=h1Int3;
                  break;

         //default:
      }
      return h0Filter;
   }//end of

   /** read analysis filter values */
   public double [] getH1(int filterType){

      switch(filterType){
         case DAUBECHIES_97:
         default:
                  currentFilter=DAUBECHIES_97;
                  h1Filter=h1Daubechies7;
                  break;
        case INTEGER_53:
                  currentFilter=INTEGER_53;

                  h1Filter=h1Int3;
                  break;

         //default:
      }
      return h1Filter;
   }//end of

   /** read synthesis filter values */
   public double [] getG0(int filterType){
      double [] g0;
      double [] h0;
      double [] h1;
      switch(filterType){
```

```
        case DAUBECHIES_97:
        default:
                currentFilter=DAUBECHIES_97;
                h0=h0Daubechies9;
                h1=h1Daubechies7;
                break;
        case INTEGER_53:
                currentFilter=INTEGER_53;
                h0=h0Int5;
                h1=h1Int3;
                break;

      }
      if(!isAlpha)
        alpha=calculateAlpha(h0, h1);
      g0=calculate_g0(alpha, h1);
      g0Filter=g0;
      return g0;
    }

    /** read synthesis filter values */
    public double [] getG1(int filterType){
      double [] g1;
      double [] h0;
      double [] h1;
      switch(filterType){
        case DAUBECHIES_97:
        default:
                currentFilter=DAUBECHIES_97;
                h0=h0Daubechies9;
                h1=h1Daubechies7;
                break;
        case INTEGER_53:
                currentFilter=INTEGER_53;
                h0=h0Int5;
                h1=h1Int3;
                break;

      }
      if(!isAlpha)
        alpha=calculateAlpha(h0, h1);
      g1=calculate_g1(alpha, h0);
      g1Filter=g1;
      return g1;
    }

/** calculate synthesis filter values */
    public double[] calculate_g0(double alpha, double [] h1){
      double [] g0;

      g0=new double[h1.length];
      int wym=g0.length-1;
      //alpha*-1,n*h1[-n]
      for(int i=0; i<=wym; i++){
        g0[i]=alpha*Math.pow(-1,indexes_h1[wym-i])*h1[wym-i];
```

```
        //System.out.println("Wynik H1, index, G0["+i+"]="+h1[wym-i]+", "+indexes_h1[wym-i]+", "+g0[i]);
        //indexes_g0[i]=-1*indexes_h1[i];
      }
      return g0;
    }

  /** calculate synthesis filter values */
    public double[] calculate_g1(double alpha, double [] h0){
      double [] g1;
      g1=new double[h0.length];
      int wym=g1.length-1;
      //alpha*-1,n*h1[-n]
      for(int i=0; i<=wym; i++){
        g1[i]=alpha*Math.pow(-1,indexes_h0[wym-i])*h0[wym-i];
        //System.out.println("Wynik H0, index, G1["+i+"]="+h0[wym-i]+", "+indexes_h0[wym-i]+", "+g1[i]);
        //indexes_g0[i]=-1*indexes_h1[i];
      }
      return g1;
    }

/** calculate alpha scaling factor – synthesis filters */
    public double calculateAlpha(double []h0, double [] h1){
      double norm1, norm2, norm3, norm4;


    switch(currentFilter){
          case DAUBECHIES_97:
                    indexes_h0=new double[h0.length];
                    indexes_h1=new double[h1.length];
                    for(int j=-4,i=0;j<=4;j++,i++)
                      indexes_h0[i]=j;
                    for(int j=-4,i=0;j<=2;j++,i++)
                      indexes_h1[i]=j;
                    break;

          case INTEGER_53:
                    indexes_h0=new double[h0.length];
                    indexes_h1=new double[h1.length];
                    for(int j=-2,i=0;j<=2;j++,i++)
                      indexes_h0[i]=j;
                    for(int j=-1,i=0;j<=1;j++,i++)
                      indexes_h1[i]=j;
                    break;
          default:
      }

      norm1=0;
      norm2=0;
      norm3=0;
      norm4=0;

      for(int i=0; i<h0.length; i++){
        norm1+=h0[i];
        norm4+=( Math.pow(-1,indexes_h0[i])*h0[i]);
      }
```

```
      for(int i=0; i<h1.length; i++){
         norm3+=h1[i];

         norm2+=( Math.pow(-1,indexes_h1[i])*h1[i]);
      }

      double norm=((norm1*norm2) +(norm3*norm4));
      if(norm==0){
         norm=2;
         //throw (new ArithmeticException("Division by zero in calculation of Alpha component for
"+currentFilter+" wavelet filter "));
      }
      isAlpha=true;
      return Math.abs(2/norm);

   }//end of calculateAlpha

   public void resetAlpha(){
      isAlpha=false;
   }//end of resetAlpha()

   public double [] sample(double []data, int freq, int start){
      int size=(int)(data.length/freq);
      double [] result = new double[size];

      for(int i=0, j=start; i<result.length; i++,j+=freq){
         result[i]=data[j];
      }
      return result;
   }

   public double [] zeroFill(double []data, int freq, int start){
      int size=(int)(freq*data.length);
      double [] result = new double[size];

      for(int i=0, j=0; i<result.length; i++){
         if(((i+start)%freq)==0)
            result[i]=0;
         else{
            result[i]=data[j];
            j++;
         }
      }
      return result;
   }


   public static void main(String args[]){

      //double data[]={10,
20,30,20,10,20,30,20,10,20,30,20,30,20,10,20,30,20,10,20,30,20,30,20,10,20,30,20,30,20,10,20,30
,20,10,20,30};
      double data[]={100, 100,100,100,200,200,200,200};
      WaveletFilters wf = new WaveletFilters();
      String h0=Arrays.toString(wf.getH0(5));
```

```
    String h1=Arrays.toString(wf.getH1(5));
    String g0=Arrays.toString(wf.getG0(5));
    String g1=Arrays.toString(wf.getG1(5));

    Convolution con = new Convolution();
    double res1[], res2[];
    res1=con.convolve1D(data,wf.getH0(5),Convolution.SYMMETRIC, false);
    String d=Arrays.toString(data);
    System.out.println("Oto dane org:"+d+"\n Oto filtr: "+h0+", "+h1);

    res2=con.convolve1D(data,wf.getH1(5),Convolution.SYMMETRIC, false);
     String r1=Arrays.toString(res1);
    String r2=Arrays.toString(res2);
    System.out.println("Oto dane: alpha="+wf.alpha+ "\n"+r1+"\n"+r2);

    res1=wf.sample(res1,2,1);
    res2=wf.sample(res2,2,0);
    r1=Arrays.toString(res1);
    r2=Arrays.toString(res2);
    System.out.println("Oto dane: sample="+"\n"+r1+"\n"+r2);

    res1=wf.zeroFill(res1,2,0);
    res2=wf.zeroFill(res2,2,1);
    r1=Arrays.toString(res1);
    r2=Arrays.toString(res2);
    System.out.println("Oto dane: zaeroFill="+"\n"+r1+"\n"+r2);

    res1=con.convolve1D(res1,wf.getG0(5),Convolution.SYMMETRIC, false);
    res2=con.convolve1D(res2,wf.getG1(5),Convolution.SYMMETRIC,false);
    r1=Arrays.toString(res1);
    r2=Arrays.toString(res2);

    System.out.println("Oto dane: alpha="+wf.alpha+ "\n"+r1+"\n"+r2);

    System.out.println("Oto dane: alpha="+wf.alpha+ "\n"+h0+"\n"+h1+"\n"+g0+"\n"+g1);

    double res[]=new double[res1.length];
    for(int l=0; l<res.length; l++){
       res[l]=res1[l]+res2[l];
    }
    r1=Arrays.toString(res);
    System.out.println("Oto wynik po rekonstrukcji"+"\n"+r1+"\n");

  }//main()

}//end of class WaveletFilters
```

// Quantization.java

```
/**
 *
 * @author  jwr
 */
```

```
public class Quantization {

   /** Creates a new instance of Quantization */
   public Quantization() {
   }

/** kwantyzacja klasyczna */
 public double [][] quantizy(double[][] data, int step){
      int h=data.length;
      int w=data[0].length;
      int v;
      // quantization step
      int d=step;
      double [][] iData = new double[h][w];
      //v=sign(y)(Math.abs(y)/d)

      for(int i=0;i<h;i++){
            for(int j=0;j<w;j++){
               iData[i][j]=Math.signum(data[i][j])*(Math.abs(Math.ceil((data[i][j])/d)));
               //System.out.print(";"+iData[i][j]);
            }
      }

      return iData;
}//end of quantizy()

/** kwantyzacja płaszczyzn bitowych */
public double [][] quantizyBP(double [][] iData, int nBits){

   int w=iData.length;
   int h=iData[0].length;
   //ile bitow przesunąć
   nBits=8-nBits;
   //double [][] qData = new int[w][h];

   for(int i=0;i<w;i++){
         for(int j=0;j<h;j++){
            iData[i][j]=(double)((((int)iData[i][j])>>nBits)<<nBits);
            //iData[i][j]=((int)iData[i][j])<<nBits;
         }
   }
   return iData;

}//end of quantizyBP()

}//class
```

// Jpeg2000Data.java

```
/**
 *  Representation of data in wavelets processing – banks, bit planes, etc.*/
 * @author  jwr
 */
```

```
public class Jpeg2000Data{

   public boolean [] BP7;
   public int noOfBanks;
   public byte[][] qData;
   public int w;
   public int h;

/** Representation of data in wavelets processing – banks, bit planes, etc.*/
   public Jpeg2000Data(byte [][] data, int nB){
         qData=data;
         w=data.length;
         h=data[0].length;
         noOfBanks=nB;
   }
   public Jpeg2000Data( int nB){
         noOfBanks=nB;
   }



public boolean[][] getBP(byte [][] data, int BPno){
   boolean [] []bp=null;
   int mask=255;
   switch(BPno){
         case 0: mask=0x1;
         case 1: mask=0x2;
         case 2: mask=0x4;
         case 3: mask=0x8;
         case 4: mask=16;
         case 5: mask=32;
         case 6: mask=64;
         case 7: mask=128;
   }
   for(int i=0; i<w;i++){
         for(int j=0;j<h;j++){
         if((qData[i][j]&mask)!=0)
               bp[i][j]=true;
         else
               bp[i][j]=false;

         }
   }
   return bp;
}//end of boolean[][] getBP(byte [][] data, int BPno){

public byte [][]getBanksData(int noBanks){
   if(noBanks>noOfBanks){
   System.out.println("Not enough data");
   return null;
   }
   int sizeW;
   int sizeH;
```

```
   if(noBanks==noOfBanks){
    sizeW=w;
    sizeH=h;
   } else{
   sizeW=w/((noOfBanks-noBanks)*2); sizeH=h/((noOfBanks-noBanks)*2);

   }

   byte [][] rData=new byte[sizeW][sizeH];
   for(int i=0; i<sizeW;i++){
         for(int j=0;j<sizeH;j++){
         rData[i][j]=qData[i][j];
         }
   }
   return rData;

}//end of Public getBanksData()

/*
 quarterNo= 1 [gorny, prawy]=2 [dolny, lewy], =3[dolny, prawy]
*/
public byte[][] getAddBanksData(int noBanks, int quarterNo){
   if(noBanks>noOfBanks){
   System.out.println("Not enough data");
   return null;
   }
   int sizeW;
   int sizeH;
   int dif=(noOfBanks-noBanks);
   if(dif==0){
    sizeW=w/2;
    sizeH=h/2;
   } else{
    sizeW=(int)(w*Math.pow(0.5,dif+1));
    sizeH=(int)(h*Math.pow(0.5,dif+1));
   }
   //w*Math.pow(0.5,dif+1);

   byte [][] rData=new byte[sizeW][sizeH];
   for(int i=0; i<sizeW;i++){
         for(int j=0;j<sizeH;j++){
            switch(quarterNo){
               case 1:
                     rData[i][j]=qData[i+sizeW][j];
                     break;
               case 2:
                     rData[i][j]=qData[i][j+sizeH];

                     break;
               case 3:
                     rData[i][j]=qData[i+sizeW][j+sizeH];

                     break;

            }//end of switch
```

```
        }//end of for j
    }//end of for i
    return rData;



}//end of Public byte[][] getAddBanksData()


public static double [][] getBank(double [][] data, int bankNo){
    if (bankNo==0) return null;
    int bankH=data.length/(int)Math.pow(2.0,(double)bankNo);  //(bankNo*2);
    int bankW=data[0].length/(int)Math.pow(2.0,(double)bankNo); //(bankNo*2);
    double [][] bankData = new double[bankH][bankW];
    for(int i=0; i<bankH; i++){
        for(int j=0; j<bankW; j++){
            bankData[i][j]=data[i][j];
        }
    }
    return bankData;
}

public static double [][] setBank(double [][] fullData, double [][] bankData){
    //if (bankNo==0) return null;
    int bankH=bankData.length;
    int bankW=bankData[0].length;
    int dataH=fullData.length;
    int dataW=fullData[0].length;
    for(int i=0; i<bankH; i++){
        for(int j=0; j<bankW; j++){
            fullData[i][j]=bankData[i][j];
        }
    }
    return fullData;

}

}//end of class Jpeg2000Data
```

// Jpeg2000Procesor.java

```
/**
 * Forward and reverse DWT using filter banks
 * @author  jwr
 */
public class Jpeg2000Procesor {

   public double[][] sData;

    public int filterType;

public Jpeg2000Procesor( int filterType){
```

```
   this.filterType=filterType;
}

public void setData(double[][] sData){
   this.sData=sData;
}



public double[][] procesIDWT(double[][] sData, int filterType, boolean rows){
      int h=sData.length;
      int w=sData[0].length;
      /*
      if ((h%2)!=0) h=h-1;
      if ((w%2)!=0) w=w-1;
       */
      //System.out.println("Oto w,k="+w+","+h);

      Convolution con=new Convolution();
      double [] hfFilter;
      double [] lfFilter;
      WaveletFilters wf= new WaveletFilters();
      lfFilter=wf.getG0(filterType);
      hfFilter=wf.getG1(filterType);
      //
      double[][] results=new double[h][w];
      if(!rows){
         int tmp=w;
         w=h;
         h=tmp;
      }

      double[] hfData=new double[w];
      double[] lfData=new double[w];
      int half=w/2;
      double [] buffer1=new double[half];
      double [] buffer2=new double[half];
      //double [] buffer1Zero=new double[w];
      //double [] buffer2Zero=new double[w];


      for(int i=0; i<h;i++){
         for(int j=0; j<w;j++){
            if(rows)
               if(j<half){
                  buffer1[j]=sData[i][j];
               } else {
                  buffer2[j-half]=sData[i][j];
               }
            else
               if(j<half){
                buffer1[j]=sData[j][i];
                } else {
                  buffer2[j-half]=sData[j][i];
               }
```

```
        }

        //res2=wf.zeroFill(res2,2,1);
        hfData=con.convolve1D(wf.zeroFill(buffer2,2,1),hfFilter,Convolution.SYMMETRIC,false);
        lfData=con.convolve1D(wf.zeroFill(buffer1,2,0),lfFilter,Convolution.SYMMETRIC,false);
        //hfData=wf.sample(hfData,2,0);
        //lfData=wf.sample(lfData,2,1);
        //System.out.println("Oto dane w,h, lfData.length"+w+", "+h+", "+lfData.length);
        for(int l=0; l<lfData.length;l++){
           if(rows){
              results[i][l]=lfData[l]+hfData[l];
              //results[i][l+lfData.length]=hfData[l];
           }else {
              results[l][i]=lfData[l]+hfData[l];
              //results[l+lfData.length][i]=hfData[l];
           }//end if rows
        }//end for l

     }//end for i
     return results;
  }

public double[][] procesDWT(double[][] sData, int filterType, boolean rows){
     int h=sData.length;
     int w=sData[0].length;
     //System.out.println("Oto w,k="+w+","+h);

     Convolution con=new Convolution();
     double [] hfFilter;
     double [] lfFilter;
     WaveletFilters wf= new WaveletFilters();

     lfFilter=wf.getH0(filterType);
     hfFilter=wf.getH1(filterType);
     //
     double[][] results=new double[h][w];
     if(!rows){
        int tmp=w;
        w=h;
        h=tmp;
     }

     double[] hfData=new double[w];
     double[] lfData=new double[w];

     double [] buffer=new double[w];

     for(int i=0; i<h;i++){
        for(int j=0; j<w;j++){
           if(rows)
              buffer[j]=sData[i][j];
           else
              buffer[j]=sData[j][i];
        }
        hfData=con.convolve1D(buffer,hfFilter,Convolution.SYMMETRIC,false);
```

```
        lfData=con.convolve1D(buffer,lfFilter,Convolution.SYMMETRIC,false);
        hfData=wf.sample(hfData,2,0);
        lfData=wf.sample(lfData,2,1);

        for(int l=0; l<lfData.length;l++){
           if(rows){
              results[i][l]=lfData[l];
              results[i][l+lfData.length]=hfData[l];
           }else {
              results[l][i]=lfData[l];
              results[l+lfData.length][i]=hfData[l];
           }//end if rows
        }//end for l

     }//end for i
     return results;
}// end of procesDWT

public void processJpeg(int noOfBanks){

         Quantization q= new Quantization();
         sData=procesDWT(procesDWT(sData,filterType,true),filterType,false);
         //sData=q.quantizy(sData);
}//processJpeg()

}//end of class
```

//Główny plik aplikacji
// Jpeg2000DemoLocal.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import java.net.*;
import java.util.Locale;
import java.io.*;



/**
 * Demonstracja dekompozycji falkowej obrazu, kwantyzacji, kompresji i dekompresji obrazów
 * @author  jwr
 */
public class Jpeg2000DemoLocal extends JFrame{
   public static final String INFO="INFO:";

   public static final int BANKS_LIMIT=5;
   public static final int GAIN=10;
   public ImagePanel imagePanel=new ImagePanel();
   public ImagePanel mrwPanel=new ImagePanel();
   public JTextArea infoArea=new JTextArea(INFO);
   public JPanel controlPanel=new JPanel();
   public JButton reconstruct=new JButton("Odtwórz");
```

```
   public JButton info=new JButton("Info");
   public JButton moreBanks=new JButton("+ poziom");
   public JButton lessBanks=new JButton("- poziom");
   public JButton openImage=new JButton("Otwórz obraz");
   public JButton quantization=new JButton("Kwantyzuj");
   public JButton quantizationBP=new JButton("KwantyzujBP");
   public JButton end=new JButton("KONIEC");
   public JTextField qStep = new JTextField("1");
   public JTextField qStepBP = new JTextField("8");
   public JTextField recBanks = new JTextField("1");
   JPanel leftPanel = new JPanel();
   JPanel rightPanel = new JPanel();
   public static File katalog=null;
   public PrintStream log=System.out;

   public int[] imageData;
   public double[] data;
   public double [][] dataTMP=null;
   public int w;
   public int h;
   public int step=1;
   public int stepBP=8;
   public JFrame parent;
   public int noOfBanks=1;
   public int banks=1;
   public boolean procesStatus = false;



   /** Creates a new instance of Jpeg2000Demo */
   public Jpeg2000DemoLocal() {
      super("Wymiana obrazow z zastosowaniem dekompozycji falkowej i kwantyzacji JPEG2000 - Jacek
Ruminski");
      parent=this;
      init();
   }

   public void init(){

      JPanel infoGroup = new JPanel();
      infoGroup.setLayout(new GridLayout(4,1));
      setLayout(new GridLayout(1,2));
      leftPanel.setLayout(new BorderLayout());
      rightPanel.setLayout(new BorderLayout());
      getContentPane().add(leftPanel);
      getContentPane().add(rightPanel);
      JPanel qant= new JPanel();
      qant.setLayout(new GridLayout(1,2));
      qant.add(new JLabel("PODAJ SKOK (MIANOW.) KWANTYZACJI:"));
      qStep.setColumns(20);
      qant.add(qStep);


      JPanel qantBP= new JPanel();
      qantBP.setLayout(new GridLayout(1,2));
```

```
qantBP.add(new JLabel("L. BITÓW (PŁASZCZ.) DO KWANTYZACJI:"));
qStepBP.setColumns(20);
qantBP.add(qStepBP);

JPanel banksP= new JPanel();
banksP.setLayout(new GridLayout(1,2));
banksP.add(new JLabel("PODAJ L. WARSTW DO REKONSTRUKCJI:"));
recBanks.setColumns(20);
banksP.add(recBanks);
leftPanel.add(imagePanel,BorderLayout.CENTER);

infoGroup.add(qant);
infoGroup.add(qantBP);
infoGroup.add(banksP);
infoGroup.add(infoArea);
leftPanel.add(infoGroup,BorderLayout.SOUTH);

rightPanel.add(mrwPanel,BorderLayout.CENTER);
rightPanel.add(controlPanel,BorderLayout.SOUTH);
controlPanel.setLayout(new GridLayout(2,4));
controlPanel.add(openImage);

controlPanel.add(moreBanks);
controlPanel.add(lessBanks);
controlPanel.add(reconstruct);
controlPanel.add(info);
controlPanel.add(quantization);
controlPanel.add(quantizationBP);
end.setForeground(Color.red);
controlPanel.add(end);

imagePanel.setPreferredSize(new Dimension(512, 512));
mrwPanel.setPreferredSize(new Dimension(512, 512));

info.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent ae){
     JOptionPane.showMessageDialog(parent,new String("Program demonstracyjny elementy JPEG2000:
\nsterowaniem dynamiką rozdzielczości przestrzennej i jakości.\n Jacek Rumiński"),new String("INFORMACJI
!"),JOptionPane.INFORMATION_MESSAGE   );

   }
 });

moreBanks.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent ae){
      if (noOfBanks< BANKS_LIMIT) {
         //if (!procesStatus){
            noOfBanks++;
            infoArea.setText("LICZBA WARSTW = "+noOfBanks);
            process();
         //}
      }

   }
 });
```

```
    lessBanks.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
        if (noOfBanks>1) {
          //if (!procesStatus) {
            noOfBanks--;
            infoArea.setText("LICZBA WARSTW = "+noOfBanks);
            process();
          //}
        }
      }
    });

    reconstruct.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
        String recBanksS = recBanks.getText();
        try{

          banks=Integer.parseInt(recBanksS);

          System.out.println("L. poz. do rek.: "+banks+", l. poz.: "+noOfBanks+", limit: "+BANKS_LIMIT);
          if (banks>noOfBanks) throw (new NumberFormatException());
          //System.out.println("Liczba BANKS: "+noOfBanks);

          if(banks>BANKS_LIMIT) throw (new NumberFormatException());
          //noOfBanks=banks;
          if (procesStatus) reconstruct();

        }catch (NumberFormatException e){
          JOptionPane.showMessageDialog(parent,new String("Wartość liczby warstw musi być liczbą
całkowitą, mniejszą od "+BANKS_LIMIT+". \n Dodatkowe informacje: "+e),new String("OSRZEŻENIE
!"),JOptionPane.WARNING_MESSAGE);
        }

      }
    });

    quantization.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
        String stepS = qStep.getText();

        try{
          step=Integer.parseInt(stepS);
          Quantization q=new Quantization();
          dataTMP=q.quantizy(dataTMP, step);

        }catch (Exception e){
          JOptionPane.showMessageDialog(parent,new String("Wartość kroku kwantyzacji musi być liczbą
całkowitą"),new String("OSRZEŻENIE !"),JOptionPane.WARNING_MESSAGE);
        }


      }
    });
```

```
quantizationBP.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent ae){

      String stepBPS = qStepBP.getText();

      try{
         stepBP=Integer.parseInt(stepBPS);
         Quantization q=new Quantization();
         dataTMP=q.quantizyBP(dataTMP, stepBP);

      }catch (Exception e){
         JOptionPane.showMessageDialog(parent,new String("Wartość kroku kwantyzacji musi być liczbą
całkowitą"),new String("OSRZEŻENIE !"),JOptionPane.WARNING_MESSAGE);
      }

   }
});


end.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent ae){
      shutdown();
   }
});

openImage.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent ae){
      URL imageFileURL=openFileDialog();
      //Image img = Toolkit.getImage(imageFileURL);
      if(imageFileURL!=null){
         Image img = Toolkit.getDefaultToolkit().getImage(imageFileURL);
         MediaTracker mt = new MediaTracker(parent);
         mt.addImage(img,0);
         if (mt.isErrorAny()) System.out.println("BŁĄD ŁADOWANIA OBRAZU");
         try { mt.waitForID(0); }
         catch (InterruptedException ie) {return;  }
         if (mt.isErrorID(0)) return;
         w = img.getWidth(null);
         h = img.getHeight(null);
         if((w>512)||(h>512)){
            JOptionPane.showMessageDialog(parent,new String("Obraz za duży skaluję do rozmiaru do
512/512"),new String("OSRZEŻENIE !"),JOptionPane.WARNING_MESSAGE);
            double scale=1;
            if(w>=h)
               scale=w/512.0;
            else
               scale=h/512.0;
            w=(int)(w/scale);
            h=(int)(h/scale);
            //PONI□EJ TYMCZASOWE USTAWIENIE - BRAK DODAWANIA LINII CZY KOLUMN
            w=512;
            h=512;
            img=img.getScaledInstance(w,h,Image.SCALE_SMOOTH);
            mt = new MediaTracker(parent);
```

```
            mt.addImage(img,0);
            if (mt.isErrorAny()) System.out.println("BŁĄD ŁADOWANIA OBRAZU");
            try { mt.waitForID(0); }
            catch (InterruptedException ie) {return;  }
            if (mt.isErrorID(0)) return;
          }

          BufferedImage bObraz = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
          Graphics2D big = bObraz.createGraphics();
          big.drawImage(img,0,0,null);
          bI_to_int(bObraz);
          resetImagePanel();
          dataTMP=null;
          addWaveletPanel(imageData);
          //ob=new Obraz(bObraz);
        }
          //openImage(imageFileURL);

    }
  });

}



private URL openFileDialog(){
  URL fileURL=null;
  boolean cancel=false;
  try{
    JFileChooser chooser = new JFileChooser();
    if (Jpeg2000Demo.katalog!=null)
        chooser.setCurrentDirectory(Jpeg2000Demo.katalog);

    chooser.setLocation(this.getWidth()+10,0);
    chooser.setBackground(Color.black);
    chooser.setDialogTitle("OTWÓRZ");
    chooser.setApproveButtonText("OTWÓRZ");
    chooser.setToolTipText("Tekst");
    chooser.setDefaultLocale(new Locale("pl","PL"));
    int returnVal = chooser.showOpenDialog(this);

    if(returnVal == JFileChooser.APPROVE_OPTION) {
        File file=chooser.getSelectedFile();
        fileURL=new URL("file","localhost",file.getAbsolutePath());
        log.println("OTO URL: "+fileURL);
    }else
         cancel=true;
    Jpeg2000Demo.katalog=chooser.getCurrentDirectory();
    log.println("KATALOG: "+Jpeg2000Demo.katalog.getAbsolutePath());
  }catch(Exception e){
    if(!cancel)
      log.println("Błąd dostępu do pliku, przy próbie jego otwarcia! Kod"+e);
  }
```

```
    return fileURL;
  }//private URL openFileDialog(){

  public static void shutdown(){
    System.exit(1);
  }//shutdown()



  /**
  * bI_to_int - metoda pobrania wartości macierzy obrazu z jego
  * reprezentacji w formie BufferedImage, do postaci macierzy luminancji
  * Y=(R+G+B)/3
  */
  public void bI_to_int(BufferedImage bOb){
        /*
        int w=bOb.getWidth();
        int h=bOb.getHeight();
        this.w=w;
        this.h=h;
         */
         log.println("Width:"+w);
         log.println("Height:"+h);
        int MAX=Integer.MIN_VALUE;
        int MIN=Integer.MAX_VALUE;
        //int [][]mob=new int[h][w];
        imageData=new int[w*h];
        data=new double[w*h];
        int pixel=0;
        int R=0,G=0,B=0;
        for(int i=0;i<w;i++){
              for(int j=0;j<h;j++){
                    pixel=bOb.getRGB(i,j);
                    B=pixel&0xFF;
                    G=(pixel>>8)&0xFF;
                    R=(pixel>>16)&0xFF;
                    //mob[j][i]=(R+G+B)/3;
                    int l=(R+G+B)/3;
                    data[j*w+i]=(double)l;
                    imageData[j*w+i]=(255 << 24)| (l << 16) |(l << 8) | l;
                    if (l>MAX) MAX=l;
                    if (l<MIN) MIN=l;
                    //if (mob[j][i]>MAX) MAX=mob[j][i];
                    //if (mob[j][i]<MIN) MIN=mob[j][i];
              }
        }

        //imageData=Macierz.macWektor(h,w,mob);
        //mob=new int[1][1];
  }// koniec bI_to_int

  public void resetImagePanel(){
    //Graphics g=imagePanel.getGraphics();
    Image img=produceImage(imageData, w,h);
```

```
    imagePanel.setImage(img);
    validate();
    repaint();

  }//end of resetImagePanel

  public void addWaveletPanel(int []imageData){
    infoArea.setText("LICZBA WARSTW = "+noOfBanks);
    process();
  }

  /**
   * Dekompozycja falkowa obrazu
   */
  public void process(){
    Jpeg2000Procesor jp = new Jpeg2000Procesor(5);
    if (dataTMP==null)
      dataTMP = new double[h][w];
    int offset=1;
    double [][] dataPart;
    //if(!procesStatus){
      for(int i=0; i<h; i++){
        offset=i*w;
        for(int j=0; j<w; j++){
          dataTMP[i][j]= data[offset+j];
          //if(i==5) System.out.print(";"+dataTMP[i][j]);
        }
      }
    //}

    for (int i=0; i<noOfBanks; i++){
      if(i>0){
        dataPart=Jpeg2000Data.getBank(dataTMP,i);
        dataPart=jp.procesDWT(dataPart,  5, true);
        dataPart=jp.procesDWT(dataPart,  5,  false);
        dataTMP=Jpeg2000Data.setBank(dataTMP,dataPart);
      } else {
        dataTMP=jp.procesDWT(dataTMP,  5, true);
        dataTMP=jp.procesDWT(dataTMP,  5,  false);
      }
    }

    int [] results = new int[w*h];
    offset=1;
    int l;
    int max=Integer.MIN_VALUE;
    int min=Integer.MAX_VALUE;
    int h_half=h/(2);
    int w_half=w/(2);
    for(int i=0; i<h; i++){
      offset=i*w;
      for(int j=0; j<w; j++){
        l=results[offset+j]=(int)dataTMP[i][j];
        if (l<min) min =l;
        if(l>max) max=l;
```

```
            if((i>h_half)||(j>w_half)){
               l=results[offset+j]=GAIN*l;
            }else {

            }

         }
      }

      double a = 255.0/(max-min);
      double b= -1*a*min;
      //a=1.0;
      //b=0.0;
      //System.out.println("Oto skal:"+a+", "+b);
      for (int i=0; i<results.length; i++){
         l=(int)(a*results[i]+b);
         results[i]=(255 << 24)| (l << 16) |(l << 8) | l;
      }

      mrwPanel.setImage(produceImage(results,w,h));
      rightPanel.validate();
      rightPanel.repaint();
      validate();
      repaint();
      procesStatus=true;
   }


   /**
    * Dekompresja - rekonstrukcja obrazu
    */
   public void reconstruct(){
      //mrwPanel=new ImagePanel();

      Jpeg2000Procesor jp = new Jpeg2000Procesor(5);
      //dataTMP = new double[h][w];
      int offset=1;
      int diff=noOfBanks-banks;

      int rH=h;
      int rW=w;
      if (diff>0) rH= (int)(h/(Math.pow(2.0,(double)(diff))));
      if (diff>0) rW= (int)(w/(Math.pow(2.0,(double)(diff))));


      double [][] dataRES = new double [rH][rW];
      for(int i=0; i<rH; i++){
         offset=i*rW;
         for(int j=0; j<rW; j++){
            dataRES[i][j]= dataTMP[i][j];
            //if(i==5) System.out.print(";"+dataTMP[i][j]);
         }
      }
      System.out.println("TEST przed IWT, banks= "+banks);
       double [][] dataPart;
```

```
      for (int i=banks; i>0; i--){
        if(i>1){
           dataPart=Jpeg2000Data.getBank(dataRES,(i-1));
           dataPart=jp.procesIDWT(dataPart,  5, false);
           dataPart=jp.procesIDWT(dataPart,  5,  true);
           dataRES=Jpeg2000Data.setBank(dataRES,dataPart);
           System.out.println("TEST nr "+i);
        } else {
              dataRES=jp.procesIDWT(dataRES,  5, false);
              dataRES=jp.procesIDWT(dataRES,  5, true);
        }
      }
      System.out.println("data:");
      int [] results = new int[rW*rH];
      offset=1;
      int l;
      int max=Integer.MIN_VALUE;
      int min=Integer.MAX_VALUE;
      for(int i=0; i<rH; i++){
         offset=i*rW;
         for(int j=0; j<rW; j++){
            l=results[offset+j]=(int)dataRES[i][j];
            //if(i==5) System.out.print(";"+dataTMP[i][j]);
            if(l>max) max=l;
            if (l<min) min =l;
         }
      }

      double a = 255.0/(max-min);
      double b= -1*a*min;
      System.out.println("Oto skal:"+a+", "+b);
      for (int i=0; i<results.length; i++){
         l=(int)(a*results[i]+b);
         l=l&0xff;
         results[i]=(255 << 24)| (l << 16) |(l << 8) | l;
      }

      mrwPanel.setImage(produceImage(results,rW,rH));
      rightPanel.validate();
      rightPanel.repaint();
      validate();
      repaint();
      procesStatus=false;
      noOfBanks=0;
   }

/** Create an Image object from matrix data */
   public Image produceImage(int [] imageData, int w, int h){

     Image img = createImage(new MemoryImageSource(w, h, imageData, 0, w));

      //tworzony jest obraz w RGB o szerokości w, wysokości h,
      //na podstawie tablicy próbek pix, bez przesunięcia w tej tablicy z w elementami w linii
      return img;
```

```
    }

    public static void main (String args[]){
        Jpeg2000DemoLocal jp = new Jpeg2000DemoLocal();
        jp. setSize(1024,768);
        jp.setVisible(true);
    }//end of main()

}//class
```