Przetwarzanie obrazów cyfrowych – laboratorium

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt „Programowa i strukturalna reforma systemu kształcenia na Wydziale Mat-Fiz-Inf”.
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Człowiek-najlepsza inwestycja

Uniwersytet Marii Curie-Skłodowskiej
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Przetwarzanie obrazów cyfrowych – laboratorium

Marcin Denkowski
Paweł Mikołajczak

Lublin 2011

Instytut Informatyki UMCS
Lublin 2011

Marcin Denkowski
Paweł Mikołajczak

Przetwarzanie obrazów cyfrowych – laboratorium

Recenzent: Michał Chlebiej

Opracowanie techniczne: Marcin Denkowski
Projekt okładki: Agnieszka Kuśmierska

Praca współfinansowana ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Publikacja bezpłatna dostępna on-line na stronach Instytutu Informatyki UMCS: informatyka.umcs.lublin.pl.

Wydawca
Uniwersytet Marii Curie-Skłodowskiej w Lublinie
Instytut Informatyki
pl. Marii Curie-Skłodowskiej 1, 20-031 Lublin
Redaktor serii: prof. dr hab. Paweł Mikołajczak
www: informatyka.umcs.lublin.pl
email: dyrii@hektor.umcs.lublin.pl

Druk
ESUS Agencja Reklamowo-Wydawnicza Tomasz Przybylak
ul. Ratajczaka 26/8
61-815 Poznań
www: www.esus.pl

ISBN: 978-83-62773-02-2

Spis treści

Przedmowa

System wzrokowy człowieka jest niezmiernie skomplikowany, dostarcza ponad 90% informacji potrzebnych mu do funkcjonowania w środowisku. Otoczenie odwzorowywane jest w postaci obrazu. Scena zapisana w obrazie jest przetwarzana, tak aby wyselekcjonować potrzebne człowiekowi informacje. Tworzenie i analizowanie obrazów towarzyszy człowiekowi od samego początku historii człowieka myślącego. Stosunkowo od niedawna (lata dwudzieste dwudziestego wieku) dostępna stała się nowa forma obrazu – obrazy cyfrowe. Obrazy cyfrowe są specyficzną formą danych. W obrazach, zarówno zarejestrowanych przez urządzenia do ich akwizycji jak i obrazach postrzeganych bezpośrednio przez system wzrokowy człowieka, występuję duża nadmiarowość. Istotne jest wyodrębnienie kluczowych cech zawartych w przedstawianym obrazie. Podstawowym zadaniem technik przetwarzania obrazów cyfrowych jest analiza obrazów, która prowadzi do uzyskania istotnych informacji niezbędnych przy podejmowaniu decyzji. Klasycznym przykładem użyteczności przetwarzania obrazów cyfrowych są techniki rozpoznawania linii papilarnych w celu identyfikacji osoby, jakie stosują służby imigracyjne USA na swoich przejściach granicznych. W celu zapewnienia bezpieczeństwa lotów prowadzona jest na lotniskach analiza obrazów pozyskanych z prześwietlania bagaży podróżnych. W dzisiejszych czasach trudno także sobie wyobrazić diagnozy lekarskie bez wykonania i analizy cyfrowych zdjęć medycznych (USG, CT, MRI). Są to wystarczające dowody, aby metody przetwarzania i analizy obrazów cyfrowych uznać za kluczowe zadanie współczesnej informatyki stosowanej.

W programach nauczania informatyki na wyższych uczelniach przedmioty związane z omawianiem metod przetwarzania obrazów cyfrowych zajmują znaczące miejsce. Istnieje bogata oferta podręczników omawiających zagadnienia związane z prezentowaniem teoretycznych podstaw tworzenia oraz przetwarzania obrazów cyfrowych. Oferta pozycji książkowych dotycząca praktycznych aspektów przetwarzania obrazów cyfrowych jest dość uboga.

Niniejszy podręcznik przeznaczony jest dla studentów informatyki specjalizujących się w dziedzinie przetwarzania obrazów cyfrowych. Jego przeznaczeniem jest służenie pomocą studentom tworzącym na ćwiczeniach la-

boratoryjnych oprogramowanie realizujące wybrane metody przetwarzania i analizy obrazów cyfrowych. Podręcznik zakłada, że czytelnik jest zaznajomiony z wybranymi środowiskami programistycznymi (C++/QT, Java, C# lub MATLAB).

Podręcznik składa się z siedmiu części.

W części pierwszej przedstawiono wybrane środowiska programistyczne, w kontekście przetwarzania obrazów. Omówiono zastosowanie pakietu QT w połączeniu z językiem C++, języka Java oraz środowiska .NET i języka C# a także omówiono skrótowo zalety pakietu MATLAB. W każdym opisie pokazano szkielet programu obsługującego wczytanie i przetwarzanie obrazu o wyspecyfikowanym formacie.

W rozdziale drugim zostały omówione zagadnienia związane z transformacjami intensywności obrazów. Omówiono transformacje liniowe, logarytmiczne i potęgowe oraz zagadnienia związane z przekształceniem histogramu obrazu.

Rozdział trzeci omawia zagadnienia związane z transformacjami geometrycznymi obrazów cyfrowych. W tym rozdziale omówiono także zagadnienia związane z interpolacją obrazów.

Rozdział czwarty poświęcony jest omówieniu modeli barw stosowanych w grafice komputerowej i obrazach cyfrowych. Dokładnie omówiono modele RGB, CMYK, HSL oraz podstawowe modele CIE.

Bardzo rozbudowany rozdział piaty poświęcony jest zagadnieniom związanych ze stosowaniem filtrów cyfrowych w przetwarzaniu obrazów. Omówiono filtrację wykorzystującą operację splotu, szczegółowo opisując praktyczne aspekty stosowania filtrów rozmywających i wyostrzających. Kolejny podrozdział omawia zagadnienia związane z filtrami nieliniowymi, koncentrując się na omówieniu filtrów statystycznych i filtrów adaptacyjnych.

W rozdziale szóstym omówiono podstawy przekształceń morfologicznych, przedstawiając podstawowe elementy teorii. W części praktycznej omówione zostały dobrze działające algorytmy morfologiczne.

Dość obszerny rozdział siódmy omawia zagadnienia związane z przetwarzaniem obrazów cyfrowych w domenie częstotliwościowej. Omówiona jest dyskretna transformata Fouriera i dyskretna transformata kosinusowa oraz zastosowanie ich przykładowych implementacji. Przedstawiono praktyczne zastosowanie transformat do obliczania szybkiego splotu oraz kompresji stratnej obrazu. Omówiono także zagadnienie osadzania znaku wodnego w obrazie cyfrowym.

Podręcznik powstał na podstawie dziesięcioletniego doświadczenia autorów podręcznika w prowadzeniu zajęć z przedmiotów grafika komputerowa, przetwarzanie obrazów cyfrowych i przetwarzanie obrazów medycznych na kierunku Informatyka na Uniwersytecie Marii Curie-Skłodowskiej w Lublinie.

Rozdział 1

Środowisko programistyczne

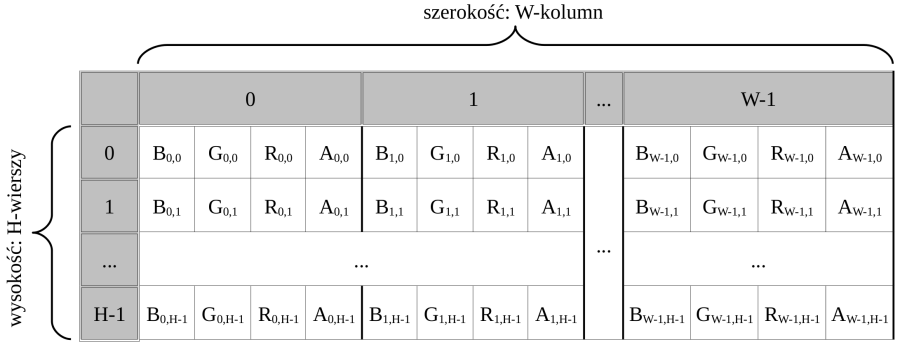
1.1. Cyfrowa reprezentacja obrazu

Obraz cyfrowy jest reprezentowany logicznie przez dwuwymiarową tablicę, macierz. Ilość wierszy i kolumn tej tablicy definiuje rozdzielczość obrazu w pionie i w poziomie. Każdy element tej macierzy opisuje wartość pojedynczego punktu obrazu - piksela. Piksel jest opisywany pewną strukturą reprezentującą poziom szarości lub kolor punktu. W zależności od wielkości tej struktury mówimy o rozdzielczości bitowej obrazu lub o jego głębi bitowej. W historii cyfrowego obrazu struktura punktu przyjmowała coraz większe wielkości :

1) 1 bit – punkt mógł mieć tylko jedną z dwóch wartości, zazwyczaj interpretowanych jako kolor biały lub czarny. Taki obraz nazywa się monochromatycznym.
2) 8 bitów – daje możliwość reprezentacji 256 stanów punktu interpretowanych jako skala szarości od czerni do bieli lub w trybie tzw. indeksowanym, gdzie każda wartość reprezentowała z góry ustalony kolor wybrany często ze znacznie większego zbioru i zebrany w tzw. tablicy LUT (LookUp Table).
3) 15 i 16 bitów – punkt jest reprezentowany w postaci złożenia 3 podstawowych barw składowych addytywnego modelu barw RGB. Dla 15 bitowej reprezentacji istnieje $2^{15} = 32768$ możliwości, po 5 bitów na każdą składową; dla reprezentacji 16 bitowej istnieje $2^{16} = 65536$ możliwości, odpowiednio 5-6-5 bitów na każdą składową RGB.
4) 24 i 32 bity – pełna reprezentacja wszystkich dostępnych kolorów. Każdy bajt tej struktury zapisany w postaci liczby całkowitej reprezentuje stan odpowiedniej składowej - czerwonej, zielonej i niebieskiej. Nieznacznym rozwinięciem 24-bitowego opisu koloru jest tryb 32-bitowy (Rysunek 1.1 ilustruje logiczne ułożenie punktów w obrazie). Dodaje on dodatkowy 8-bitowy kanał zwany często kanałem alfa, wykorzystywany do reprezentacji przezroczystości, chociaż bardzo często po prostu ignorowany. Jest to jednak często pewnien kompromis pomiędzy wydajnością a ilością zajmowanej pamięci, bowiem struktura 4-bajtowa w 32-bitowych procesorach jest znacznie lepiej segmentowalna od 3-bajtowej.

24-bitowa liczba opisująca kolor daje w sumie dosyć imponującą liczbę możliwych do zakodowania barw, bo aż 16777216, ale należy pamiętać, że na każdą składową przypada wciąż tylko 256 możliwych stanów, co w przypadku subtelnych przejść tonalnych może się okazać niewystarczające. Warto pamiętać również, że kilka efektów i przekształceń zadanych obrazowi opisanemu liczbami całkowitymi może dosyć szybko doprowadzić do degradacji informacji w nim zawartych. Doprowadziło to do wprowadzenia reprezentacji zmiennoprzecinkowej pojedynczej precyzji lub nawet podwój-

nej precyzji. Tak dokładnie obraz reprezentują wewnętrznie współczesne karty graficzne lub zaawansowane programy do przetwarzania obrazów z najbardziej znanym Photoshopem na czele. Dalsza część niniejszej pozycji skupia się w zasadzie tylko na rozdzielczości 24/32 bitowej, wspominając w razie potrzeby o innych możliwościach.



Rysunek 1.1. Schemat logicznego ułożenia bajtów RGB w 32-bitowym obrazie w architekturze x86.

Sam model RGB w zapisie 32-bitowym jest opisany 4 bajtową liczbą, taką że najmłodszy bajt ma wartość składowej niebieskiej, drugi bajt składowej zielonej, trzeci bajt składowej czerwonej, a najstarszy bajt składowej alfa. W zapisie szesnastkowym taka liczba jest dosyć prosta do skonstruowania i zazwyczaj zapisuje się w postaci 8 cyfr heksadecymalnych:

0xAARRGGBB

gdzie AA to bajt kanału alfa, a kolejne pary liczb to składowe RGB koloru. Liczbę taką składa się ze składowych za pomocą odpowiednich sum i przesunięć bitowych, np. w notacji języka c:

```
unsigned int color = (A<<24) + (R<<16) + (G<<8) + B;
```

Natomiast w odwrotną stronę, czyli wyłuskanie składowych z koloru można zrealizować w języku c w następujący sposób:

```
unsigned char B = (color      ) & 0xFF;
unsigned char G = (color >> 8 ) & 0xFF;
unsigned char R = (color >> 16) & 0xFF;
unsigned char A = (color >> 24) & 0xFF;
```

W kolejnych podrozdziałach zostaną przedstawione podstawowe struktury i funkcje umożliwiające tworzenie i modyfikacje obrazów oraz ich prezentację na formularzu w oknie programu dla każdego z czterech języków/środowisk. Niestety ramy książki nie pozwalają na gruntowne omówienie idei budowy

aplikacji i interfejsu użytkownika. Autorzy zakładają, że czytelnik ma już pewne doświadczenie i wiedzę w pracy z wybranym środowiskiem.

1.2. C++/Qt

Język C/C++ pomimo dosyć mocnego promowania nowych technologii trzyma się mocno i pewnie jeszcze dosyć trudno będzie go zdetronizować. Sam język oczywiście w żaden sposób nie wspiera obrazów cyfrowych zatem posłużymy się biblioteką oferującą możliwość budowy interfejsu graficznego i w dużej mierze zwalniającą programistę z niskopoziomowych problemów przetwarzania obrazów. Nasz wybór to biblioteki, a w zasadzie już platforma *Qt* w wersji 4 (wymowa org. *kjut*) stworzona przez Trolltech a przejęta stosunkowo niedawno i aktywnie rozwijana przez fińską Nokię. Niewątpliwą zaletą biblioteki Qt jest jej wieloplatformowość i przenośność na poziomie kodu źródłowego. Pod tym względem stanowi ona zdecydowanie lepszą alternatywę niż rozwiązania Microsoftu.

1.2.1. Źródła w sieci

Biblioteka do celów niekomercyjnych jest dostępna na licencji LGPL na stronach Nokii pod adresem:

http://qt.nokia.com

Dostępne jest również zintegrowane środowisko programistyczne (IDE) o nazwie *Qt Creator* ułatwiające tworzenie zaawansowanych projektów. Zawsze najświeższa dokumentacja API dostępna jest pod adresem:

http://doc.qt.nokia.com

Umiejętne korzystanie z tej dokumentacji jest warunkiem koniecznym do używania Qt. Pozycje [23, 3] stanowią doskonałe wprowadzenie dla początkującego programisty Qt.

1.2.2. Kolor i obraz w Qt

Pomimo, że w Qt kolor jest reprezentowany przez klasę `QColor` dla niskopoziomowych przekształceń wydajniejszym rozwiązaniem będzie typ `QRgb`, który jest ekwiwalentem typu **`unsigned int`**:

```
1 typedef unsigned int QRgb;
```

oraz funkcji operujących na nim:

```
1 int qAlpha ( QRgb rgba );
2 int qBlue  ( QRgb rgb );
3 int qGreen ( QRgb rgb );
4 int qRed   ( QRgb rgb );
```

```
5 QRgb qRgb  ( int r, int g, int b );
6 QRgb qRgba ( int r, int g, int b, int a );
```

Pierwsze cztery pozycje pozwalają na wyłuskanie wartości konkretnej składowej koloru, a pozycje 5 i 6 ułatwiają składanie pełnego koloru z pojedynczych składowych.

Właściwy obraz w Qt jest reprezentowany przez klasę `QImage`. Daje ona możliwość opisywania koloru za pomocą 1-, 8-, 16-, 24- i 32-bitów w przeróżnych wariantach. O głębi koloru decydujemy w chwili konstruowania obiektu tej klasy za pomocą konstruktora:

```
QImage(int width, int height, Format format);
```

gdzie parametry `width i height` definiują rozdzielczość poziomą i pionową obrazu a parametr `format` typu wyliczeniowego opisuje format koloru. Najistotniejsze będą dwie jego wartości:

1) `QImage::Format_RGB32` – obraz jest 32-bitowy, ale ostatni oktet ma zawsze wartość 255 (0xFFRRGGBB)
2) `QImage::Format_ARGB32` – pełny obraz 32-bitowy (0xAARRGGBB).

Bardzo przydatny jest również konstruktor wczytujący obraz z pliku:

```
QImage(QString fileName, const char* format=0);
```

Jeżeli parametr `format` pozostanie domyślny to loader spróbuje zgadnąć na podstawie nagłówka rzeczywisty format pliku graficznego. Głębia obrazu zostanie identyczna z głębią pliku graficznego. W przypadku istniejącego już obiektu posługując się metodą:

```
bool QImage::load(QString fileName, const char* format=0);
```

można wczytać plik `fileName` zastępując tym samym wszystkie dane przechowywane w obiekcie nowo wczytanymi. Analogicznie do zapisu służy metoda:

```
bool QImage::save(QString fileName, const char* format=0,
                  int quality=-1);
```

W tym przypadku jeżeli nie poda się formatu pliku klasa spróbuje zgadnąć format na podstawie rozszerzenia. Opcjonalny parametr `quality` może mieć wartości -1 lub wartość z zakresu [0..100] i jest brany pod uwagę jedynie przy formatach umożliwiających kompresję.

Do konwersji pomiędzy formatami można wykorzystać metodę:

```
QImage QImage::convertToFormat(Format format);
```

i zdać się na wbudowane algorytmy konwersji lub oczywiście - napisać własne.

Dostęp do danych pikseli zawartych w obrazie można uzyskać w dwojaki sposób. Pierwszy, prostszy polega na użyciu metod set/get:

```
void QImage::setPixel(int x, int y, unsigned int value);
QRgb QImage::pixel(int x, int y);
```

Niestety funkcje te są bardzo kosztowne i przy transformacji wymagającej dużej wydajności nie sprawdzają się. Dużo efektywniejszym wyborem jest użycie metod:

```
char* QImage::bits();
char* QImage::scanLine(int i);
```

Pierwsza metoda jest odpowiednikiem dla wywołania `scanLine(0)`. W obu przypadkach metoda zwraca wskaźnik ustawiony na początek podanej w parametrze linii. Typ wskaźnika można w razie potrzeby rzutować na odpowiednią strukturę opisującą piksel. Typowy przykład wypełnienia całego obrazu konkretnym kolorem może wyglądać tak:

Listing 1.1. Wypełnienie obrazu zadanym kolorem.

```
1  QImage image(500, 600, QImage::Format_RGB32);
2  QRgb color = qRgb(255, 128, 64);
3  for(int y=0; y<image.height(); y++)
4  {
5      QRgb* p = (QRgb*)image.scanLine(y);
6      for(int x=0; x<image.width(); x++)
7      {
8          p[x] = color;
9      }
10 }
```

Kolejność pętli, czyli iteracja najpierw po liniach a wewnątrz po kolumnach ma spore znaczenie wydajnościowe. Klasa `QImage` przechowuje dane obrazu w jednowymiarowej tablicy o wielkości (wysokość × szerokość × ilość_bajtów_na_piksel) kolejnymi liniami obrazu, czyli w bajcie o 1 dalszym niż ostatni bajt pierwszej linii jest pierwszy bajt drugiej linii. Warto przy tym pamiętać, że metoda `scanLine()` zwraca wskaźnik do pamięci zatem nie może on być użyty poza właściwym obszarem pamięci, na którym znajdują się dane obrazu a pamiętając, że jest to obszar ciągły i jednowymiarowy w najprostszym przypadku należy tylko pilnować czy nie próbujemy zapisać jakiejś informacji przed pierwszą i za ostatnią linią obrazu. Mając to na uwadze Listing 1.1 można napisać w prostszy sposób:

Listing 1.2. Wypełnienie obrazu zadanym kolorem.

```
QImage image(500, 600, QImage::Format_RGB32);
QRgb color = qRgb(255, 128, 64);
QRgb* p = (QRgb*)image.bits();
for(int i=0; i<image.height()*image.width(); i++)
    p[i] = color;
```

Jeszcze prostszym sposobem wypełnienia obrazu jest użycie metody:

```
void QImage::fill(unsigned int pixelValue);
```

1.2.3. Zarys aplikacji

Zgodnie z regułami biblioteki Qt plik z funkcją główną w najprostszej postaci może wyglądać następująco:

Listing 1.3. Plik main.cpp.

```
#include <QApplication>
#include "imagewidget.h"

int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    ImageWidget iw;
    iw.show();
    app.exec();
}
```

Trzon aplikacji stanowi obiekt `app` klasy `QApplication` – to ten obiekt jest odpowiedzialny za działanie GUI i model zdarzeniowy aplikacji Qt. Klasa `ImageWidget` jest klasą użytkownika zdefiniowaną na listingu 1.4 oraz 1.5.

Listing 1.4. Plik imagewidget.h

```
#include <QWidget>
#include <QImage>

class ImageWidget : public QWidget
{
    QImage image;
protected:
    virtual void paintEvent(QPaintEvent*);
public:
    ImageWidget(QWidget parent=0);
};
```

Klasa `ImageWidget` dziedziczy z klasy `QWidget` i będzie stanowić formularz, na którym zostanie narysowany obraz. Obraz `image` jest prywatną składową tej klasy. Metoda `paintEvent()` to metoda wirtualna wywoływana przez system zawsze gdy niezbędne jest narysowanie bądź odrysowanie formularza. Potrzeba jeszcze definicji metod:

Listing 1.5. Plik imagewidget.cpp

```
1 #include "imagewidget.h"
2 #include <QPainter>
3
4 ImageWidget::ImageWidget(QWidget parent) : QWidget(parent)
5 {
6     image = QImage(640, 480, QImage::Format_RGB32);
7     image.fill(0x5d6d6d);
8 }
9
10 ImageWidget::paintEvent(QPaintEvent*)
11 {
12     QPainter painter(this);
13     painter.drawImage(0, 0, image)
14 ;
15 }
```

W konstruktorze tworzony jest obiekt klasy `QImage` o 640 kolumnach, 480 wierszach i 32-bitowej głębi. Obraz jest w całości wypełniany kolorem granatowym. Metoda `paintEvent()` w swoim ciele tworzy obiekt typu `QPainter` i skojarza go z własną klasą poprzez podanie wskaźnika `this` do konstruktora obiektu. Klasa `QPainter` jest główną klasą rysującą w Qt. Tu została wykorzystana do narysowania obrazu `image` na formularzu, dzięki wywołaniu metody `QPainter::drawImage(int x, int y, QImage& image)`. Parametry `x` i `y` definiują gdzie na formularzu będzie lewy, górny róg obrazu `image`.

Do kompilacji programu składającego się z tych trzech plików wystarczą 3 polecenia wydane z konsoli:

- `#qmake -project` – na podstawie plików źródłowych tworzony jest plik projektu o rozszerzeniu .pro.
- `#qmake` – z pliku projektu tworzony jest Makefile zawierający wszystkie niezbędne zależności i reguły.
- `#make` – klasyczny make wykonujący polecenia zawarte w Makefile.

1.3. Java

Java jako język wręcz stworzony do nauki programowania obiektowego a przy tym w wysokim stopniu przenośny na poziomie bajtkodu oferuje wszystko co potrzeba do implementacji algorytmów przetwarzania obrazów

oraz budowy solidnego interfejsu użytkownika. Istnieje przy tym spora ilość dobrych środowisk programistycznych, jak choćby NetBeans czy Eclipse. Istnieje również wiele podręczników do nauki zarówno języka jak i całego środowiska obiektowego z doskonałym *"Thinking in Java"* Bruce'a Eckela [12] na czele. W pozycji [5] czytelnik znajdzie szerokie omównienie algorytmów przetwarzania obrazów z przykładami implementacji w Javie.

1.3.1. Klasy koloru i obrazu w Javie

Kolor w Javie jest reprezentowany przez klasę `java.awt.Color`. Szereg konstruktorów tej klasy pozwala tworzyć jej instancję na wiele sposobów:

```
1 Color(float r, float g, float b)
2 Color(float r, float g, float b, float a)
3 Color(int r, int g, int b)
4 Color(int r, int g, int b, int a)
5 Color(int rgba)
```

W czterech pierwszych przypadkach obiekt jest tworzony na podstawie składowych R,G,B(,A). Dla typu `float` definiującego składową musi się ona zawierać w przedziale [0.0..1.0]. Dla typu `int` wartość składowych jest ograniczona do przedziału [0..255]. W obu przypadkach jeżeli wartości składowych będą poza ustalonym zakresem zostanie rzucony wyjątek `IllegalArgumentException`. Ostatni wyróżniony konstruktor tworzy obiekt koloru na podstawie 4-bajtowego zapisu koloru RGB. Dostęp do wartości składowych zapewniają akcesory:

```
1 int getRed();
2 int getGreen();
3 int getBlue();
4 int getRGB();
```

Operowanie klasą `Color` w sytuacji gdzie wymagana jest duża wydajność nie jest zbyt optymalnym rozwiązaniem. Zdecydowanie lepiej sprawdzi się zwykły typ `int` i funkcje pomocnicze do składania/rozkładania koloru:

Listing 1.6. Funkcje pomocnicze składania/rozkładania koloru w Javie.

```
1 int jrgb(int r, int g, int b){
2   return (r<<16) + (g<<8) + b;
3 }
4
5 int jred(int rgb){
6   return (byte)((rgb>>16) & 0xff);
7 }
8 int jgreen(int rgb){
9   return (byte)((rgb>>8) & 0xff);
```

```
10 }
11 int jblue(int rgb){
12   return (byte)(rgb & 0xff);
13 }
```

Do reprezentacji obrazu wykorzystamy klasę:

`java.awt.image.BufferedImage`

dziedziczącą z bardziej ogólnej `java.awt.Image`. Jej konstruktor

```
1  BufferdImage(int width, int height, int imageType);
```

pozwala stworzyć obiekt obrazu o rozmiarze `width`× `height` i typie opisanym parametrem `imageType`. Możliwe typy obrazu są zdefiniowane jako statyczne pola klasy `BufferedImage` z najistotniejszymi:

– `BufferedImage.TYPE_BYTE_GRAY` – jednobajtowy obraz w skali szarości, nieindeksowany
– `BufferedImage.TYPE_BYTE_INDEXED` – jednobajtowy obraz indeksowany,
– `BufferedImage.TYPE_INT_RGB` – czterobajtowy obraz z 8-bitowymi składowymi RGB, bez kanału alfa,
– `BufferedImage.TYPE_INT_ARGB` – czterobajtowy obraz z 8-bitowymi składowymi ARGB.

Wczytywanie obrazu z pliku graficznego jest realizowane za pomocą klasy `javax.imageio.ImageIO`:

Listing 1.7. Wczytywanie obrazu z pliku.

```
1 try {
2     BufferedImage img;
3     img = ImageIO.read(new File("file.jpg"));
4 }
5 catch (IOException e) {}
```

Zpis do pliku graficznego można zrealizować za pomocą metody `write()` klasy `ImageIO`:

Listing 1.8. Wczytywanie obrazu z pliku.

```
1 try {
2     File outputfile = new File("saved.png");
3     ImageIO.write(img, "png", outputfile);
4 } catch (IOException e) {}
```

Obsługiwane typy plików graficznych w danym systemie można uzyskać za pomocą statycznych metod klasy `ImageIO`:

```
String[] getReaderFormatNames();
String[] getWriterFormatNames();
```

Java standardowo koduje i dekoduje typowe formaty: BMP, JPG, PNG, GIF.

Dostęp do poszczególnych punktów obrazu można zrealizować za pomocą akcesorów:

```
int getRGB(int x, int y);
void setRGB(int x, int y, int rgb);
```

Jednakże wiąże się to z dużym narzutem czasowym na ewentualne konwersje modeli kolorów i testowanie współrzędnych. W przypadku przekroczenia możliwych wartości x i y rzucany jest wyjątek `ArrayOutOfBoundsException`. Lepszym rozwiązaniem jest wykorzystanie klasy `java.awt.image.Raster` i jej podklasy `java.awt.image.WritableRaster` do uzyskania bezpośredniego dostępu do tablicy danych obrazu:

Listing 1.9. Dostęp bezpośredni do punktów obrazu w JAvie.

```
1 try {
2   int rgb[] = ((DataBufferInt)img.getRaster().
3                     getDataBuffer()).getData();
4   for (int i=0; i<img.getHeight()*img.getWidth(); i++) {
5     rgb[i] = 0xff88aa;
6   }
7 } catch (Exception e) {}
```

1.3.2. Zarys aplikacji

Cała aplikacja zostanie zamknięta w pojedynczym pliku zawierającym definicję klasy dziedziczącą z klasy formularza `javax.swing.JFrame` i zawierającą funkcję główną `Main()`.

Listing 1.10. Kod aplikacji.

```
1  import java.awt.*;
2  import java.awt.image.*;
3  import java.io.*;
4  import javax.imageio.ImageIO;
5  import javax.swing.*;
6
7  public class Main extends JFrame {
8
9    private BufferedImage img = null;
10
11   public Main() {
```

```
12      setDefaultCloseOperation(
13        WindowConstants.DISPOSE_ON_CLOSE);
14
15      img = new BufferedImage(256, 256,
16                BufferedImage.TYPE_INT_RGB);
17
18      try {
19        int rgb[] = ((DataBufferInt)img.getRaster().
20                          getDataBuffer()).getData();
21        int ysh = 0;
22        for (int y = 0; y < img.getHeight(); y++) {
23          ysh = y*img.getWidth();
24          for (int x = 0; x < img.getWidth(); x++) {
25            rgb[x+ysh] = jrgb(x-y, 255-y, x^y);
26          }
27        }
28      } catch (Exception e) { System.out.println(e); }
29
30      try {
31        File outputfile = new File("saved.png");
32        ImageIO.write(img, "png", outputfile);
33      } catch (IOException e) {}
34    }
35
36    @Override
37    public void paint(Graphics g) {
38      g.drawImage(img, 10, 30, null);
39    }
40
41    public static void main(String[] args) {
42      Main main = new Main();
43      main.setVisible(true);
44      main.setSize(280, 300);
45    }
46  }
```

Linie 1-5 – niezbędne importy do działania aplikacji.

Linia 9 – deklaracja obiektu składowego typu `BufferedImage` przechowującego dane obrazu.

Linie 11-34 – definicja konstruktora klasy głównej.

Linie 12 – zmiana domyślnego zachowania w przypadku zamknięcia aplikacji.

Linia 15 – stworzenie nowego obiektu typu `BufferedImage` o 32-bitowej głębi i rozdzielczości 256×256.

Linie 18-28 – sposób uzyskania bezpośredniego dostępu do punktów obrazu za pomocą klasy `Raster`, generujący obraz ze swoistą, kolorową kratą.

W linii 25 wykorzystana jest funkcja `jrgb()` zdefiniowana na Listingu 1.6 do składania koloru ze składowych RGB.

Linie 30-34 – zapis wygenerowanego obrazu do pliku o nazwie "saved.png".

Linie 37-39 – redefinicja wirtualnej metody `void paint(Graphics)` zdefiniowanej w przodku klasy `JFrame` – `java.awt.Component`. Ta metoda jest wywoływana przez system w momencie gdy zachodzi potrzeba narysowania/odrysowania komponentu. Samo rysowanie obrazu na komponencie realizuje metoda

```
1 void drawImage(Image img, int x, int y, ImageObserver o)
```

klasy `java.awt.Graphics`.

Linie 41-45 – w funkcji głównej programu tworzony i wyświetlany jest formularz klasy `Main`.

1.4. C#

Produkt Microsoftu pomimo znacznie ograniczonej przenośności aplikacji z racji popularności systemu operacyjnego jest równie popularną platformą programistyczną co Java. W tym podrozdziale wykorzystamy biblioteki .NET w połączeniu z językiem C#. Najwygodniejszym środowiskiem do programowania w C# jest Visual Studio produkcji Microsoftu. Niestety nie jest ono darmowe. Istnieje jednak kilka wolno dostępnych rozwiązań, jak chociażby SharpDevelop pod Windows czy opensourcowy projekt Mono razem z IDE o nazwie MonoDevelop pod Linuxa.

1.4.1. Klasy obrazu i koloru w .NET

Klasą reprezentującą kolor w bibliotekach .NET jest klasa `Color` zdefiniowana w przestrzeni nazw `System.Drawing`. Wśród sporej ilości metod definiujących kolory z nazwy jest statyczna metoda:

```
Color FromArgb(int red, int green, int blue);
```

oraz jej czteroparametrowy odpowiednik:

```
Color FromArgb(int alpha, int red, int green, int blue);
```

umożliwiające składanie koloru z trzech/czterech składowych RGB. Mając tak zdefiniowany kolor można wyłuskać z niego wartość heksadecymalną koloru dzięki metodzie:

```
int ToArgb();
```

oraz poszczególne składowe przy użyciu akcesorów:

```
byte B {get;}
byte G {get;}
byte R {get;}
byte A {get;}
```

Mimo wszystko, tak jak w poprzednich przypadkach wydajniejsze będzie bezpośrednie składanie/rozkładania koloru przy użyciu funkcji bardziej niskopoziomowych, np:

Listing 1.11. Funkcje pomocnicze składania/rozkładania koloru w C#.

```
1  int csrgb(byte r, byte g, byte b){
2    return (r<<16) + (g<<8) + b;
3  }
4
5  byte csred(int rgb){
6    return (byte)((rgb>>16) & 0xff);
7  }
8  byte csgreen(int rgb){
9    return (byte)((rgb>>8) & 0xff);
10 }
11 byte csblue(int rgb){
12   return (byte)(rgb & 0xff);
13 }
```

Obraz w bibliotekach .NET reprezentuje klasa `Bitmap` zdefiniowana również w przestrzeni nazw `System.Drawing`. Najistotniejsze konstruktory tej klasy to:

```
Bitmap(String fileName);
Bitmap(Int32 width, Int32 height, PixelFormat);
```

W pierwszym przypadku stworzona zostanie instancja klasy i załadowany zostanie obraz z pliku o nazwie `fileName`. Standardowo mogą to być pliku w formacie BMP, GIF, JPG, PNG oraz TIFF. W przypadku problemów z wczytaniem pliku zostanie rzucony wyjątek `FileNotFoundException`. W drugim przypadku konstruktor inicjuje instancję `Bitmapy` nadając jej odpowiedni rozmiar i format. Akceptowane formaty zdefiniowane są w typie wyliczeniowym `PixelFormat` z przestrzeni nazw `System.Drawing.Imaging` a najistotniejszymi wartościami będą:

– `PixelFormat.Format24bppRgb` - trzybajtowy obraz z 8-bitowymi składowymi RGB,
– `PixelFormat.Format32bppRgb` - czterobajtowy obraz z 8-bitowymi składowymi RGB, czwarty bajt nie używany,

– `PixelFormat.Format32bppArgb` - czterobajtowy obraz z 8-bitowymi składowymi ARGB; zamiennie można użyć formatu `PixelFormat.Canonical`.

Z ciekawostek możliwe jest zdefiniowanie wprost głębi 48 lub 64 bitowej, gdzie każda składowa jest opisana 16 bitową liczbą całkowitą za pomocą formatów:

```
PixelFormat.Format48bppRgb,
PixelFormat.Format64bppArgb.
```

Wczytywanie obrazów z pliku jest realizowane za pomocą odpowiedniego konstruktora, natomiast zapis pliku graficznego przy użyciu metody klasy `Bitmap`:

```
void Save(string fileName)
```

lub

```
void Save(string fileName, ImageFormat format);
```

Klasa `System.Drawing.Imaging.ImageFormat` definiuje szereg własności określających typ zapisywanego pliku.

Dostęp do punktów w obrazie, znów można realizować na dwa sposoby. Za pomocą akcesorów:

```
Color GetPixel(int x, int y);
void SetPixel(int x, int y, Color color)
```

Również w tym przypadku nie jest to wydajne rozwiązanie. Zdecydowanie lepszym rozwiązaniem będzie bezpośrednia manipulacja danymi obrazu, chociaż jest to nieznacznie bardziej skomplikowane w porównaniu do C++ i Javy. Po pierwsze za pomocą metody:

```
BitmapData LockBits(Rectangle rect, ImageLockMode mode,
                    PixelFormat format)
```

klasy `Bitmap`, należy zablokować w pamięci istniejącą bitmapę. Metoda ta zwraca obiekt klasy `BitmapData`, która jest swoistym uchwytem do danych bitmapy. Parametr `rect` określa prostokątny obszar obrazu, którego część ma zostać zablokowana. Parametr `mode` jest typem wyliczeniowym określającym sposób dostępu do danych, który może mieć następujące wartości:

```
ImageLockMode.ReadOnly,
ImageLockMode.WtriteOnly,
ImageLockMode.ReadWrite.
```

Parametr `format` jest identyczny jak format bitmapy używany w konstrukto-

rze. Mając już uchwyt do danych w postaci obiektu klasy `BitmapData` można już wprost, korzystając z własności `Scan0` uzyskać wskaźnik do danych. Trzeba jednak wcześniej oznaczyć fragment kodu operujący na wskaźnikach jako `unsafe`, pamiętając, że równocześnie cały projekt staje się "unsafe".

1.4.2. Zarys aplikacji

Najprostsza aplikacja wyświetlająca obraz będzie składała się z pliku głównego oraz pliku definiującego formularz.

Listing 1.12. Plik główny programu.

```
1 using System;
2 using System.Windows.Forms;
3
4 namespace poc {
5     internal sealed class Program {
6     [STAThread]
7     private static void Main(string[] args) {
8         Application.EnableVisualStyles();
9         Application.SetCompatibleTextRenderingDefault(
10                                                  false);
11         Application.Run(new MainForm());
12     }
13     }
14 }
```

Plik ten został wygenerowany automatycznie przez środowisko IDE i jako taki nie wymaga na tym poziomie ingerencji programisty.

Listing 1.13. Plik klasy formularza.

```
1 using System;
2 using System.Drawing;
3 using System.Drawing.Imaging;
4 using System.Windows.Forms;
5
6 namespace poc {
7 public partial class MainForm : Form {
8
9     private System.Drawing.Bitmap img;
10
11     public MainForm(){
12         InitializeComponent();
13
14         this.Paint += new System.Windows.Forms.
15                             PaintEventHandler(this.iPaint);
16
17         img = new Bitmap(640, 480,
18                     PixelFormat.Format32bppRgb);
```

```
19
20          Color cl = Color.FromArgb(255, 128, 64);
21          for(int y=0; y<img.Height; y++)
22              for(int x=0; x<img.Width; x++)
23                  img.SetPixel(x,y, cl );
24
25          Rectangle rect = new Rectangle(0, 0,
26                                  img.Width, img.Height);
27          BitmapData bdata = null;
28
29          try{
30              bdata = img.LockBits(rect,
31                      ImageLockMode.WriteOnly,
32                      PixelFormat.Format32bppRgb);
33
34              unsafe {
35                  IntPtr ptr = bdata.Scan0;
36                  int* p = (int*)ptr.ToPointer();
37                  for(int i=0; i<img.Height*img.Width/3; i++)
38                      p[i] = csrgb(55,128,64);
39              }
40          } finally {
41               img.UnlockBits(bdata);
42          }
43      }
44
45      private void iPaint(object sender, PaintEventArgs e) {
46          e.Graphics.DrawImage(img , 10, 10);
47      }
48  }
49  }
```

Ten plik definiuje klasę formularza `MainForm`.

Linia 14 – w konstruktorze klasy następuje skojarzenie zdarzenia `OnPaint` formularza z odpowiednią metodą. W tym przypadku przy zdarzeniu `OnPaint` odrysowującym formularz zostanie wywołana metoda `iPaint()`.

Linia 17 – konstruowana jest tu nowa bitmapa o wielkości 640×480 i 32-bitowej głębi kolorów.

Linie 20-23 – użycie akcesora `SetPixel` do wypełnienia bitmapy zadanym kolorem.

Linie 25-43 - sposób blokowania bitmapy i użycia własności `Scan0` do uzyskania wskaźnika do danych obrazu. Cała operacja na wskaźnikach musi być zawarta w sekcji `unsafe`.

Linia 41 – odblokowanie bitmapy.

Linie 45-47 – metoda `void iPaint()` będzie wywoływana na zdarzenie `OnPaint` formularza i jedynym jej zadaniem jest narysowanie na formularzu bitmapy za pomocą metody:

```
1 void DrawImage(Image, int x, int y)
```

klasy `Graphics` przekazanej w parametrze wywołania metody.

1.5. MATLAB

MATLAB jest raczej interaktywnym środowiskiem do prowadzenia obliczeń naukowych i symulacji inżynierskich niż środowiskiem programistycznym ogólnego przeznaczenia. Jednakże dzięki bogatej bazie dodatków, tzw. toolbox-ów bardzo dobrze sprawdza się w wielu innych dziedzinach. W poniższym rozdziale wykorzystamy środowisko Matlaba rozszerzone o Image Processing Toolbox – dodatek z bogatą bazą funkcji do przetwarzania obrazów. Doskonałe wprowadzenie do środowiska MATLAB i pakietu IPT czytelnik znajdzie w pozycji [17].

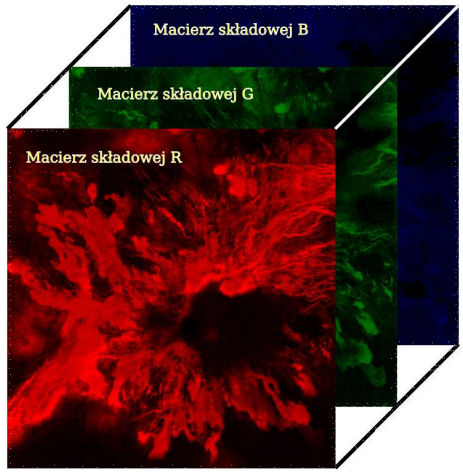
Obraz w Matlabie jest reprezentowany przez 2-wymiarową tablicę (macierz) w przypadku skali szarości i obrazu indeksowanego lub 3-wymiarową tablicę w przypadku obrazu kolorowego, a stworzenie zmiennej obrazu sprowadza się do wydania polecenia:

```
1 image = zeros(480, 640);
2 image = zeros(480, 640, 3);
3 image = uint8(zeros(480, 640, 3));
```

W pierwszym przypadku tworzona jest macierz o 640 kolumnach i 480 wierszach, pojedynczy punkt będzie opisany pojedynczą liczbą zmiennoprzecinkową podwójnej precyzji. W drugim przypadku punkt będzie opisany 3 liczbami zmiennoprzecinkowymi. Trzeci przypadek tworzy obraz o trzech składowych koloru opisanych jednobajtową liczbą całkowitą bez znaku. W każdym przypadku wartości obrazu są inicjowane zerami. Warto tutaj wspomnieć, że w Matlabie obraz może być reprezentowany za pomocą liczb 1 bajtowych całkowitych oraz liczb zmiennoprzecinkowych o podwójnej precyzji. Dla liczb całkowitych dopuszczalne wartości punktu zawierają się w zbiorze [0..255], dla liczb zmiennoprzecinkowych z zbiorze [0.0..1.0].

Dodatkową różnicą odróżniającą Matlaba od poprzednich środowisk jest sposób organizacji obrazów kolorowych. O ile w poprzednich podrozdziałach obraz kolorowy miał składowe RGB przeplatane o tyle w Matlabie jest zastosowany system płaszczyznowy, tzn. kolejne składowe RGB obrazu są zawarte w trzecim wymiarze. Taki obraz składa się z trzech macierzy o wielkości obrazu, pierwszej zawierającej wartości czerwonej składowej, drugiej zawierającej wartości zielonej składowej i ostatniej z niebieską składową. Ilustruje to Rysunek 1.2

Wczytanie pliku graficznego realizuje funkcja:



Rysunek 1.2. Ilustracja reprezentacji obrazu w Matlabie.

```
image = imread('filename');
```

Funkcja ta obsługuje większość popularnych formatów plików graficznych oraz sporą ilość bardziej egzotycznych. Do zapisu wygenerowanego obrazu służy funkcja:

```
imwrite(image, 'filename', format);
```

Parametr `format` jest łańcuchem znakowym definiującym typ pliku graficznego. Można go pominąć – wtedy funkcja zgaduje typ pliku na podstawie rozszerzenia nazwy pliku.

Za wyświetlanie obrazu odpowiada rozbudowana funkcja, która w najprostszej postaci przyjmuje tylko jeden parametr:

```
imshow(image);
```

Natomiast dostęp do wartości punktów jest realizowany w typowy dla Matlaba sposób. Listing 1.14 ilustruje kilka sposobów zarówno odczytywania wartości punktów jak i zapis do nich.

Listing 1.14. Plik klasy formularza.

```
1 r = image(15, 15, 1)
2 g = image(15, 15, 2)
3 b = image(15, 15, 3)
4 rgb = image(15, 15, 1:3)
5 subimage = image(10:15, 20:30, 1)
6 image(10, 20, 3) = 128
```

```
7 image(15:20, 10:15, 1:3) = 255*ones(6, 6, 3)
```

Linie 1-3 – odczytywane są wartości składowych RGB z punktu (15,15) do pojedynczych zmiennych.
Linia 4 – wartości RGB punktu (15,15) zebrane są w 3-elementową tablicę.
Linia 5 – z obrazu `image` wycinany jest fragment obrazu od 10 do 15 wiersza i od 20 do 30 kolumny z kanału niebieskiego i zapisywany jest do zmiennej `subimage`.
Linia 6 – zmiana składowej niebieskiej punktu (10, 15) na wartość 128.
Linia 7 – w wiersze od 15 do 20 i kolumny od 10 do 15 wpisywany jest kolor biały.

Iteracja po punktach obrazu może mieć postać:

Listing 1.15. Plik klasy formularza.

```
  image = zeros(256,256,3,'uint8');
2 for x = 1:size(image,2)
    for y = 1:size(image,1)
4     image(y,x,1:3) = [256-x,y,bitxor(y,x)];
    end
6 end
```

Najpierw tworzony jest obraz RGB o rozdzielczości 256×256 i ośmiobitowej głębi wypełniony zerami. Dwie pętle **for** iterują po wszystkich punktach obrazu tworząc w linii 4 swoistą kolorową mozaikę.

I na koniec ważna uwaga: Matlab numeruje tablice od 1 w odróżnieniu do poprzednich języków programowania.

Rozdział 2

Transformacje intensywności obrazów

Przekształcenia intensywności są najprostszymi i najbardziej podstawowymi technikami przetwarzania obrazów. Niech wartość punktu przed przekształceniem będzie oznaczona v a wartość po przekształceniu v'. Wtedy każde przekształcenie intensywności można zapisać w formie:

$$v' = T(v) \tag{2.1}$$

gdzie T będzie operatorem, który odwzorowuje wartość v na v' w określony sposób. W przypadku dyskretnym, a taki charakter mają obrazy cyfrowe, zwykle takie odwzorowanie przechowuje się w tablicy wyszukiwania określanej jako LUT (ang. *lookup table*) . Dla 8-bitowego obrazu tablica LUT będzie zawierała 256 odwzorowań liczby całkowitej z zakresu [0..255] na inną liczbę całkowitą z tego samego zakresu.

Podstawowymi przekształceniami intensywności będą funkcje: liniowe (negatyw, jasność), logarytmiczne i potęgowe (funkcja Gamma). Funkcję odcinkami liniową stosuje się przy korekcji kontrastu funkcji. Do modyfikacji kontrastu z powodzeniem stosuje się również przekształcenia histogramu obrazu.

2.1. Podstawowe przekształcenie intensywności

2.1.1. Transformacja liniowa

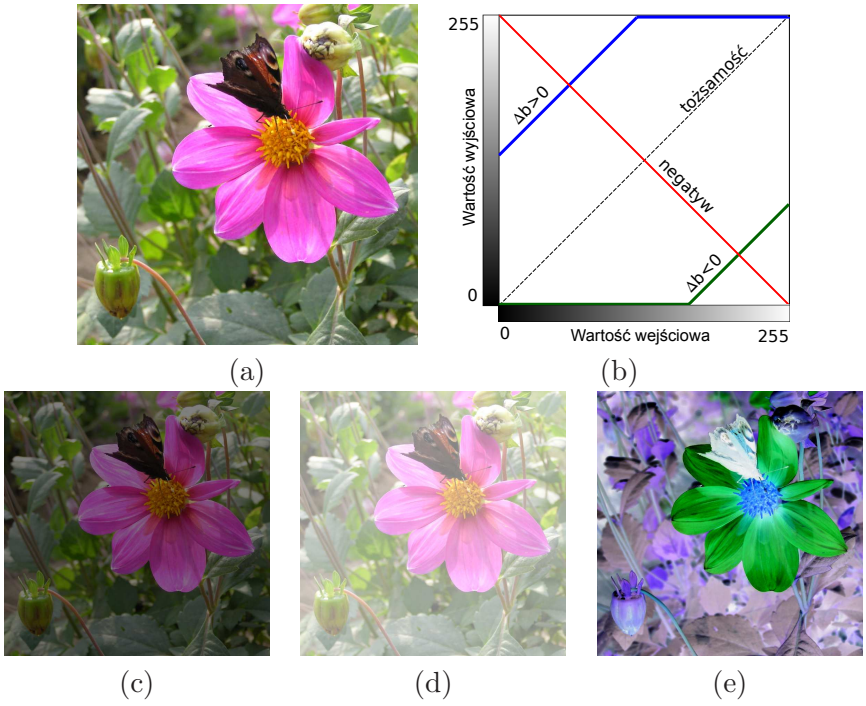
Korekcja liniowa jasności dodaje stałą wartość Δb do każdej składowej punktu :

$$v'(x, y) = a \cdot v(x, y) + \Delta b \tag{2.2}$$

gdzie a decyduje dodatkowo o kącie nachylenia prostej odwzorowania, Δb jest wartością zmiany jasności w zakresie $[-255..255]$, $v(x, y)$ jest wartością wejściową punktu o współrzędnych (x, y). Dla obrazów kolorowych, trzykanałowych w modelu RGB dla obliczenia nowej wartości punktu wyrażenie 2.2 stosuje się do każdej składowej R, G, B osobno. W obrazach 8-bitowych wszystkie wartości zawierają się w zakresie [0..255] a wszelkie operacje arytmetyczne muszą być wykonywane z nasyceniem. Trzeba zatem zadbać, żeby żadna składowa nigdy nie przekroczyła tego zakresu. W przypadku gdyby tak wynikało z obliczeń należy tą wartość przyciąć do najbliższej wartości granicznej.

Efektem zastosowania takiej transformacji przy współczynniku $a = 1$ będzie globalna zmiana jasności obrazu: rozjaśnienie gdy $\Delta b > 0$ i przyciemnienie gdy $\Delta b < 0$. Ilustruje to Rysunek 2.1.

W przypadku, gdy $a = -1$ i $\Delta b = 255$ następuje odwrócenie wartości dając w efekcie ekwiwalent fotograficznego negatywu.



Rysunek 2.1. Modyfikacja liniowa jasności obrazu barwnego: (a) obraz oryginalny, (b) funkcja transformacji liniowej, (c) obraz przyciemniony $\Delta b = -127$, (d) obraz rozjaśniony $\Delta b = 127$, (e) obraz negatywu.

2.1.2. Transformacja potęgowa

W przypadku operatora T wyrażonego wzorem :

$$T(x) = cx^{\gamma} \tag{2.3}$$

gdzie c i *gamma* są dodatnimi stałymi mamy do czynienia z tzw. korekcją gamma. Dziedziną tej funkcji jest przedział domknięty [0..1]. Wartość wykładnika potęgi decyduje o wizualnym efekcie przekształcenia, który można podzielić na trzy kategorie: (1) dla $0 < \gamma < 1$ funkcja odwzorowuje niewielką ilość ciemnych wartości na większą liczbę wartości, dając w wyniku obraz rozjaśniony, (2) dla $\gamma = 1$ mamy przekształcenie tożsame, które nie powoduje żadnej zmiany w obrazie, (3) dla $\gamma > 1$ funkcja odwzorowuje niewielką ilość jasnych wartości na większą liczbę wartości, powodując przyciemnienie obrazu. Przykładowe funkcje gamma przedstawione są na Rysunku 2.2.

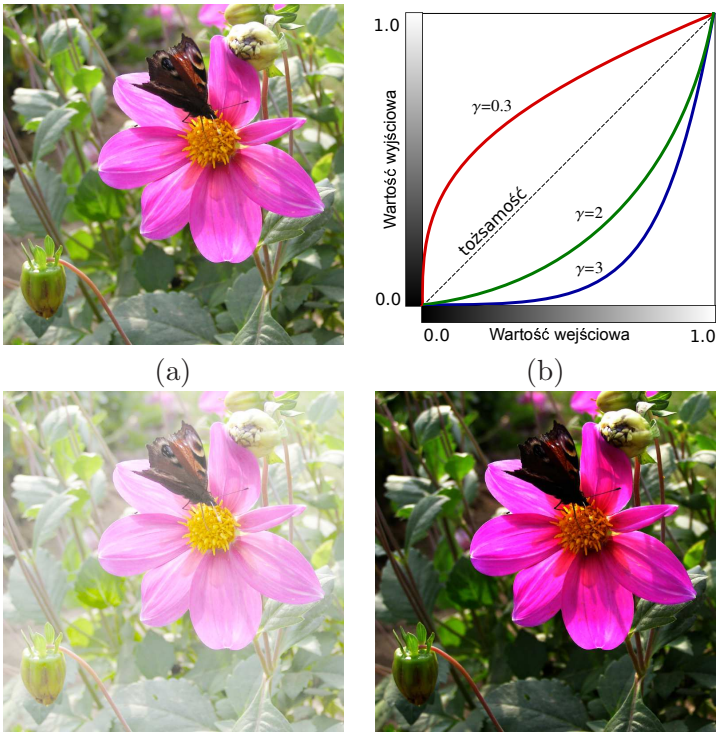
Korekcja gamma odgrywa bardzo dużą rolę w tzw. preprocesingu obrazów, ponieważ wiele urządzeń używanych do przechwytywania obrazu lub jego wyświetlania ma charakterystykę potęgową. Na przykład, klasyczny ekran z kineskopem katodowym (CRT) ma odwzorowanie

intensywność-na-napięcie, które jest funkcją typowo potęgową z wykładnikiem zawierającym się zwykle w przedziale (1.8..2.5). Z tego też powodu bez odpowiedniego przygotowania obrazu urządzenie takie produkowałoby obraz ciemniejszy niż obraz zamierzony. Zatem obraz cyfrowy, tuż przed wyświetleniem trzeba poddać korekcji gamma z odpowiednim odwróconym wykładnikiem ($\frac{1}{2.5}..\frac{1}{1.8}$). Podobne działanie należy podjąć w przypadku innych urządzeń, jak skanery czy drukarki [33].

W przypadku obrazów cyfrowych przekształcenie gamma można zapisać w postaci:

$$v'(x,y) = v(x,y)^{\gamma} \tag{2.4}$$

przyjmując stałą $c = 1$. Należy pamiętać, że wartość $v(x,y)$ w równaniu 2.4 jest wartością składowej koloru unormowaną do jedności, czyli zawiera się w zakresie [0.0..1.0], zatem, dla obrazów 8-bitowych, wartość wejściową należy podzielić przez 255 a wynik potęgowania pomnożyć przez 255.



(a) (b)

(c) (d)

Rysunek 2.2. Transformacja potęgowa obrazu: (a) obraz oryginalny, (b) funkcje korekcji gamma, (c) obraz z korekcją $\gamma = 0.33$, (d) obraz z korekcją $\gamma = 2.0$.

Korekcja gamma sprawdza się również w klasycznym przetwarzaniu obrazów do rozjaśniania lub przyciemniania. W przeciwieństwie do liniowej funkcji jasności nie powoduje tak dużej utraty informacji dodatkowo wpływając również na kontrast obrazu.

Do obliczenia wyniku potęgi można użyć funkcji `double std::pow(double x, double y)` ze standardowej biblioteki matematycznej (deklaracja w pliku `<cmath>`) w przypadku języka C++.

Dla języka Java można posłużyć się statyczną metodą klasy `java.lang.Math`: `public static double pow(double a, double b)`.

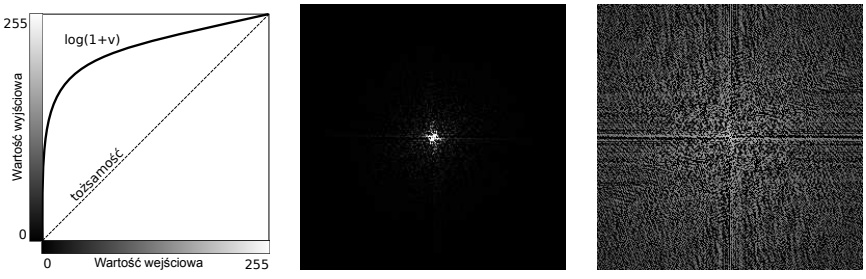
Język C# oferuje podobną metodę:
`public static double Pow(double x, double y)` klasy `System.Math`.

2.1.3. Transformacja logarytmiczna

Ogólna forma operatora przekształcenia logarytmicznego ma postać :

$$T(x) = c\log(1+x) \tag{2.5}$$

gdzie c jest stałą a $x \geqslant 0$. Kształt krzywej logarytmicznej przedstawiony na Rysunku 2.3 pokazuje, że tego typu przekształcenie odwzorowuje wąski przedział intensywności na znacznie szerszy przedział, zawęża natomiast przedział wysokich intensywności. Funkcja ta mimo podobnego działania do funkcji potęgowej nie jest tak uniwersalna. Jednakże odwzorowanie logarytmiczne ma jedną istotną właściwość: kompresji bardzo dużej rozpiętości wartości intensywności w obrazie. Przykładowe zastosowanie tej funkcji będzie dyskutowane w rozdziale 7 przy okazji wizualizacji transformaty Fouriera.



(a) (b) (c)

Rysunek 2.3. Transformacja logarytmiczna obrazu: (a) funkcja transformacji, (a) obraz o dużej rozpiętości tonalnej o charakterystyce logarytmicznej, (c) obraz po korekcji.

2.1.4. Kontrast

Jako kontrast obrazu cyfrowego będziemy rozumieć różnicę między jego intensywnościami niskimi a wysokimi, punktami ciemnymi a jasnymi. Obraz o niskim kontraście zawiera się w niewielkim przedziale możliwych wartości z całego zbioru wartości dostępnego do danego typu urządzenia obrazującego. Analogicznie, obraz o wysokim kontraście będzie zawierał wartości z całego dostępnego przedziału intensywności.

Rozważmy transformację odcinkami liniową pokazaną na Rysunku 2.4(a) używaną zwykle do modyfikacji kontrastu obrazu. Punkty (s_1, t_1) oraz (s_2, t_2) kontrolują kształt funkcji transformującej. W przypadku gdy $s_1 = t_1$ oraz $s_2 = t_2$ wtedy uzyskana transformacja jest przekształceniem tożsamym. W przypadku gdy $s_1 = s_2$ oraz $t_1 = 0$ i $t_2 = L$ gdzie L jest maksymalną możliwą wartością intensywności, wtedy uzyskana transformacja jest tzw. funkcją progującą (ang. *thresholding function*), która generuje obraz monochromatyczny, bitowy. W pozostałych przypadkach istotne są jeszcze dwa typy ułożeń punktów (s, t): (1) gdy punkt o współrzędnych (s_1, t_1) jest poniżej funkcji tożsamej a punkt (s_2, t_2) powyżej obraz będzie bardziej kontrastowy od swojego pierwowzoru, (2) gdy punkt o współrzędnych (s_1, t_1) jest powyżej funkcji tożsamej a punkt (s_2, t_2) poniżej obraz będzie mniej kontrastowy od swojego pierwowzoru,

Dla uproszczenie stosuje się zwykle tylko jedną liczbę do opisania zmiany kontrastu zamiast czterech współrzędnych. Rozważmy dwie wersje takiej liniowej korekcji kontrastu. W obu przypadkach wartości punktu dzielimy na ciemne i jasne. Ciemne to te, które mają wartości mniejsze od połowy wartości maksymalnej (dla obrazów 8-bitowych 127) a jasne te te które mają wartości większe lub równe.

1. W tym przypadku kontrast uzyskujemy poprzez zmianę kąta nachylenia prostej na wykresie odwzorowania kolorów (Rysunek 2.4(b)):

$$(a)\ \ v'(x,y) = \frac{127}{127-\Delta c}\left(v(x,y) - \Delta c\right), \quad \text{zwiększanie kontrastu} \tag{2.6}$$

$$(b)\ \ v'(x,y) = \frac{127+\Delta c}{127} v(x,y) - \Delta c, \quad \text{zmiejszenie kontrastu} \tag{2.7}$$

gdzie Δc jest wartością kontrastu w zakresie [0..127] dla przypadku (a) i $\Delta c \in [-128..0)$ dla przypadku (b).

2. W tym przypadku również zmieniany jest kąt nachylenia ale osobno dla jasnych i osobno dla ciemnych punktów (Rysunek 2.4(c)):

$$(a)\ v'(x,y) = \begin{cases} \frac{127-\Delta c}{127}v(x,y) & \text{dla } v(x,y) < 127 \\ \frac{127-\Delta c}{127}v(x,y) + 2\Delta c & \text{dla } v(x,y) \geqslant 127 \end{cases} \quad (2.8)$$

$$(b)\ v'(x,y) = \begin{cases} \frac{127}{127+\Delta c}v(x,y) & \text{dla } v(x,y) < 127 + \Delta c \\ \frac{127v(x,y)+255\Delta c}{127+\Delta c} & \text{dla } v(x,y) > 127 - \Delta c \\ 127 & \text{dla pozostałych} \end{cases} \quad (2.9)$$

Przypadek (a) zwiększa kontrast obrazu dla parametru $0 < \Delta c < 127$. Przypadek (b) zmniejsza kontrast obrazu dla parametru $-127 < \Delta c < 0$.

2.2. Przekształcenia histogramu

Histogram cyfrowego obrazu, którego możliwe wartości intensywności zawarte są w przedziale $[0..L-1]$ jest dyskretną funkcją:

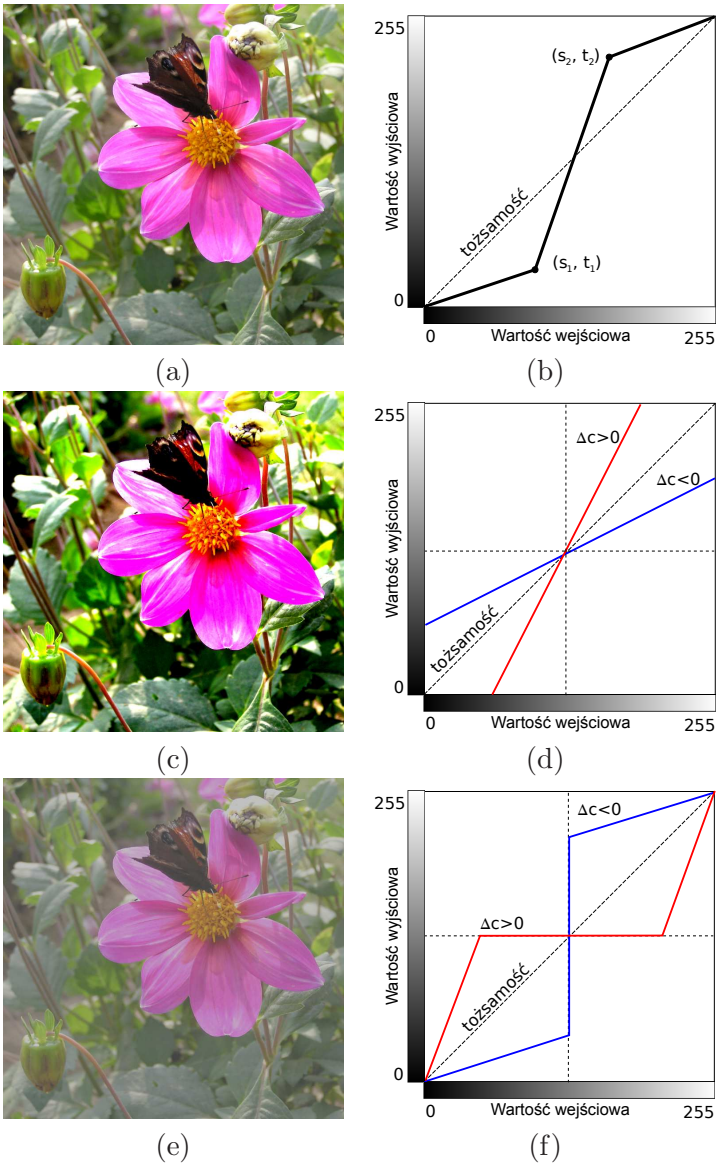
$$h(i_k) = n_k \quad (2.10)$$

gdzie i_k jest $k-$tą wartością intensywności a n_k jest ilością punktów w obrazie o intensywności i_k. Często histogram normalizuje się poprzez podzielenie każdej jego składowej przez ilość punktów w obrazie, określonej jako iloczyn NM, gdzie M i N są wielkościami rozdzielczości poziomej i pionowej obrazu. Zatem histogram może być określony jako:

$$h_N(i_k) = n_k/MN \quad (2.11)$$

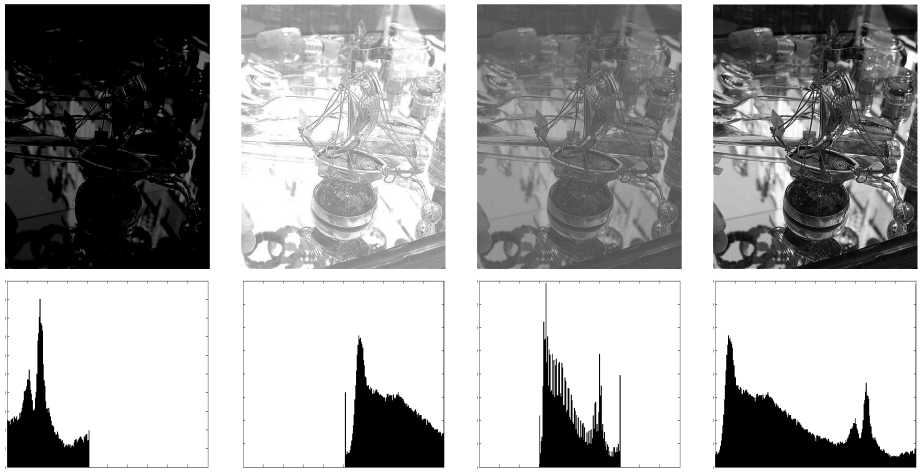
W tak znormalizowanym histogramie suma wszystkich jego składowych jest równa 1. Inaczej mówiąc histogram jest rozkładem prawdopodobieństwa wystąpienia wartości intensywności i_k w obrazie. Rozważmy obrazy pokazane na Rysunku 2.5 przedstawiające cztery charakterystyki intensywności obrazów: (a) ciemny, (b) jasny, (d) o niskim kontraście, (d) o wysokim kontraście oraz odpowiadające im histogramy. Histogramy są przedstawione w postaci wykresu, na którym oś pozioma odpowiada wartości intensywności i_k, natomiast oś pionowa odpowiada wartości $k-$tej składowej histogramu $h(i_k) = n_k$ lub $h_N(i_k) = n_k/MN$ dla znormalizowanych wartości histogramu.

Dla przypadku (a) obrazów ciemnych składowe niezerowe histogramu koncentrują się po lewej stronie skali intensywności, dla obrazu (b) jasnego



Rysunek 2.4. Modyfikacja kontrastu obrazu barwnego: (b) ogólna funkcja transformacji, (d) pierwszy uproszczony sposób korekcji kontrastu, (f) drugi uproszczony sposób korekcji kontrastu, (a) obraz oryginalny, (c) obraz o zwiększonym kontraście sposobem pierwszym $\Delta c = 50$, (e) obraz o zmniejszonym kontraście sposobem drugim $\Delta c = -50$.

analogicznie po prawej stronie skali intensywności. Obraz o niskim kontraście (c) będzie miał histogram, którego składowe będą ulokowane w centrum



(a) (b) (c) (d)

Rysunek 2.5. Przykładowe obrazy oraz ich histogramy dla obrazów: (a) ciemnych, (b) jasnych, (c) o niskim kontraście, (d) o wysokim kontraście.

skali intensywności. Im węższy będzie histogram tym obraz będzie mniej kontrastowy. Histogram obrazu o dużym kontraście (d) będzie zajmował całą dostępną przestrzeń intensywności $[0..L-1]$. Co więcej rozkład intensywności punktów jest dosyć jednorodny, tzn. większość "słupków" histogramu jest podobnej wysokości. Obraz o takim histogramie będzie wysoko kontrastowy z dużą ilością różnorodnych odcieni szarości. Większość dyskutowanych w tym podrozdziale technik przekształcania histogramu będzie miała na celu właśnie takie przekształcenie intensywności punktów obrazu, w wyniku którego histogram będzie jak najbardziej szeroki i jednorodny.

Z histogramu można również uzyskać kilka istotnych wielkości statystycznych opisujących obraz. Średnia wartość intensywności histogramu wyrażona jako:

$$\overline{h} = \sum_{k=0}^{L-1} i_k \cdot h_N(i_k) \tag{2.12}$$

będzie równa średniej wartości intensywności liczonej dla całego obrazu. Wariancja intensywności obliczona na podstawie histogramu wyrazi się wzorem:

$$\sigma^2 = \sum_{k=0}^{L-1} (i_k - \overline{h})^2 \cdot h_N(i_k) \tag{2.13}$$

Tak zdefiniowana wariancja (czyli kwadrat standardowego odchylenia) jest miarą kontrastu obrazu.

Przedstawiona tu dyskusja dotyczy obrazów w skali szarości. Dla obrazów kolorowych najprostszym rozwiązaniem jest przedstawianie osobnych histogramów dla każdej składowej koloru. (Możliwe jest do wyobrażenia obliczenie histogramu łącznego dla wszystkich składowych. Taki histogram byłby trójwymiarową tablicą o wielkości $256 \times 256 \times 256$ przechowującą w poszczególnych komórkach ilości wektorów o trzech składowych (r_i, g_i, b_i). Nie byłby to jednak twór zbyt pomocny przy przetwarzaniu obrazu nie wspominając o jego wizualizacji.) Dla modelu RGB często przedstawia się jeszcze dodatkowy histogram intensywności punktów obliczonych jako średnia ważona poszczególnych składowych przedstawiona w rozdziale 4 o równaniu 4.2 lub 4.3. Osobnym problemem jest przekształcanie histogramu obrazów kolorowych. Tutaj również można przekształcać każdy kanał koloru osobno. Jednakże, jeżeli istotne jest zachowanie proporcji pomiędzy składowymi koloru, lepsze efekty da przekształcanie histogramu poziomów szarości i modyfikacja każdej składowej koloru o taką samą zmianę intensywności szarości. Niestety, problem zachowania proporcji w takim przekształceniu jest niebanalny. Jednym z prostszych przykładów tego typu przekształcenia może być modyfikacja każdej składowej o procentową zmianę poziomu szarości. Na przykład, w modelu RGB trzy składowe $[r, g, b]$ dają intensywność równą v, ta intensywność ulega przekształceniu histogramowemu $T(h)$ dając nową wartość v'. Ta wartość jest następnie odwzorowywana na nowe wartości składowe $[r', g', b']$:

$$[r, g, b] \mapsto v \xrightarrow{T(h)} v' \mapsto [r' = \frac{v'}{v} r, g' = \frac{v'}{v} g, b' = \frac{v'}{v} b] \tag{2.14}$$

2.2.1. Wyrównywanie histogramu

Wyrównywanie histogramu (ang. *histogram equalization*) jest metodą modyfikacji intensywności punktów obrazu, który po transformacji będzie miał pożądany kształt histogramu. Technika ta używa nieliniowego odwzorowania, które tak przekształca intensywności punktów, że wyjściowy histogram ma jednorodny rozkład intensywności, innymi słowy jest płaski w kontekście gęstości prawdopodobieństwa. W przypadku funkcji ciągłej histogramu, którego całka na określonym obszarze $[0..L-1]$ jest równa 1 operacja wyrównania wygeneruje funkcję stałą i równą $\frac{1}{L-1}$ na tym obszarze:

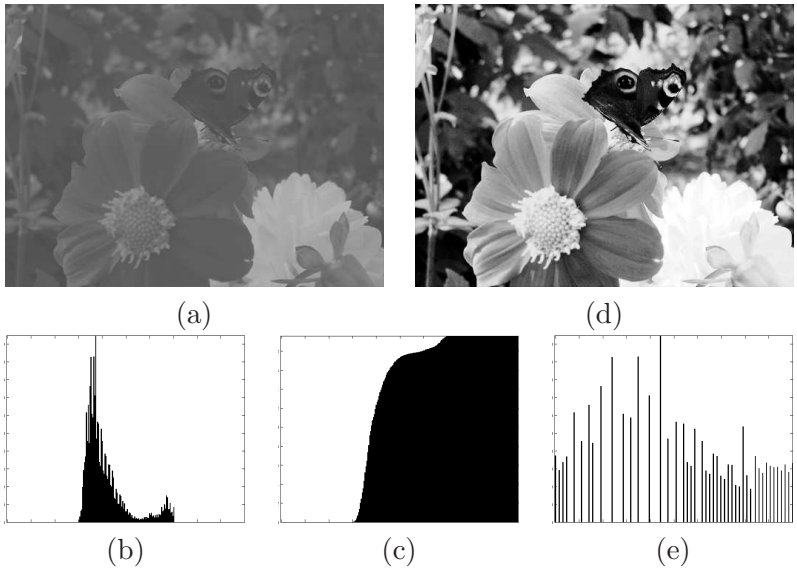
$$s = T(i) = (L-1) \int_0^i h_N(w) dw \tag{2.15}$$

Prawa strona tego równania jest nazywana kumulacyjną funkcją rozkładu.

Dla dyskretnych wartości, jakimi posługujemy się w reprezentacji obrazu i jego histogramu, funkcja transformacji zastępuje całkowanie sumowaniem a wartości funkcji wartościami prawdopodobieństwa:

$$s_k = T(i_k) = (L-1)\sum_{w=0}^{k} h_N(i_w) = \frac{L-1}{NM}\sum_{w=0}^{k} n_w \tag{2.16}$$

W ten sposób każdy punkt danego obrazu źródłowego o intensywności i_w jest mapowany na wartość s_k w obrazie wyjściowym. Rysunek 2.6 obrazuje procedurę wyrównywania histogramu dla obrazu w skali szarości.



Rysunek 2.6. (a) Obraz o niskim kontraście poddany procedurze wyrównania histogramu, (b) histogram obrazu (a), (c) histogram kumulacyjny, (d) obraz po wyrównaniu histogramu i (e) jego histogram.

Wyrównywanie histogramu dla dyskretnej postaci histogramu jak widać nie powoduje jego spłaszczenia w sensie dosłownym a odwzorowuje składowe histogramu (słupki) w ten sposób, że większe składowe oddalają się od sąsiednich składowych a mniejsze przybliżają się do siebie. Innymi słowy, wartości często występujące w obrazie zwiększają różnicę intensywności pomiędzy sobą, stąd też wrażenie poprawy kontrastu obrazu. Co więcej w rezultacie zbiór nowych wartości intensywności pokrywa cały dostępny zakres skali szarości obrazu. Po wyrównaniu ilość składowych w histogramie jest zawsze nie większa niż ilość składowych w histogramie obrazu źródłowego.

Zakładając, że wszystkie obliczenia są wykonywane w dziedzinie liczb całkowitych, procedura wyrównywania histogramu może składać się z następujących kroków:

1. Obliczenie histogramu h dla obrazu.
2. Obliczenie histogramu kumulacyjnego h_c na podstawie histogramu h za pomocą rekurencyjnego wyrażenia:

$$\begin{aligned} h_c[0] &= h[0] \\ h_c[i] &= h[i] + h_c[i-1] \quad \text{dla } 1 \leqslant i \leqslant 255 \end{aligned}$$

3. Wyznaczenie dla każdego punktu obrazu nowej wartości intensywności $v'(x, y)$:

$$v'(x, y) = \frac{255}{NM} \cdot h_c[v(x, y)] \tag{2.17}$$

gdzie $v(x, y)$ jest intensywnością punktu obrazu źródłowego, N i M są rozdzielczością poziomą i pionową obrazu.

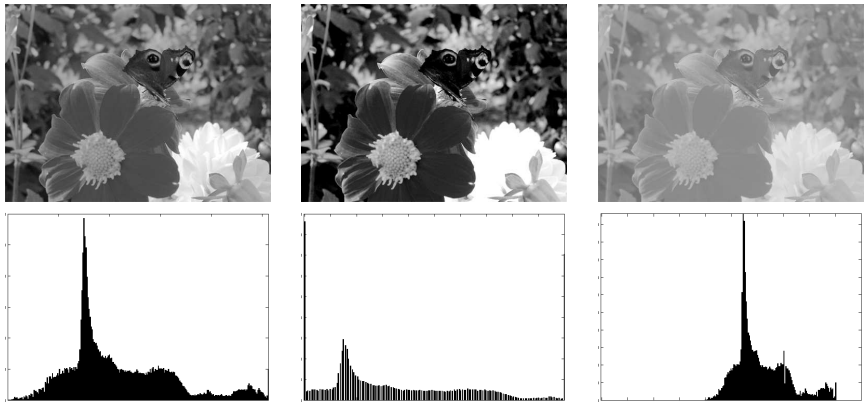
2.2.2. Skalowanie histogramu

Skalowanie histogramu (rozszerzanie kontrastu, ang. *histogram streching, levels*) jest techniką liniowej zmiany jasności punktów poprzez przeskalowanie zakresu intensywności do pewnych ustalonych granic. Do określenia zostaje zakres intensywności, który ma być przeskalowany i zakres na jaki ma być przeskalowany. Załóżmy, że c i d określają zakres docelowy skalowania (często przyjmuje się pełen zakres intensywności, tzn. $c = 0$ i $d = 255$), a a i b określają zakres który ma być skalowany. Przy tak określonych granicach skalowania rozszerzanie kontrastu będzie się wyrażać następującym wzorem:

$$v'(x, y) = (v(x, y) - a) \cdot \frac{d - c}{b - a} + c \tag{2.18}$$

gdzie $v(x, y)$ jest wartością punktu obrazu źródłowego a $v'(x, y)$ wartością obrazu wyjściowego. Zwyczajowo zmianę granic a i b przy ustalonych c i d nazywa się zmianą poziomów wejściowych (ang. *input levels*) a efektem tej zmiany jest zwiększenie kontrastu obrazu. Zmianę granic wyjściowych c i d przy ustalonych a i b nazywa się zmianą poziomów wyjściowych(ang. *output levels*) i powoduje ona zmniejszenie globalnego kontrastu obrazu. Ustalenie sensownych granic wymaga pewnej wprawy i/lub eksperymentów. Automatyczne procedury rozszerzania kontrastu ustalają granice $c = 0$ i $d = 255$ a granice a i b na podstawie najmniejszej i największej intensywności punktu w obrazie lub wartości o 5% większe i mniejsze odpowiednio. Inną możliwością ustalenia granic a i b jest wyznaczenie największej składowej histogramu i na jej podstawie zdefiniowanie pewnego poziomu odcięcia, poniżej którego

wartości histogramu będą ignorowane. Wtedy granica a będzie ustalona poprzez skanowanie histogramu od lewej strony, aż do składowej, która będzie miała wartość większą niż ustalony poziom odcięcia. Analogicznie granica b ustalana będzie przez skanowanie histogramu od strony prawej do lewej. Rysunek 2.7 pokazuje rezultaty skalowania histogramu dla poziomów wejściowych i wyjściowych.



(a) (b) (c)

Rysunek 2.7. (a) obraz oryginalny wraz z histogramem, (b) obraz o zwiększonym kontraście poprzez modyfikację poziomów wejściowych oraz jego histogram, parametry wejściowe: $a = 50, b = 205$, (c) obraz o zmniejszonym kontraście poprzez modyfikację poziomów wyjściowych oraz jego histogram, parametry wyjściowe: $c = 102, d = 230$.

2.3. Tryby mieszania obrazów

Tryb mieszania (ang. *blending modes*) dwóch obrazów jest techniką łączenia obrazów poprzez odpowiednie wyliczanie wartości ich składowych [52]. Rozważmy łączenie dwóch obrazów obrazów przedstawionych na Rysunku 2.8.

Przyjmijmy, że a jest wartością składowej koloru podstawowego a b wartością składowej koloru mieszanego. W każdym rozważanym przypadku trybu mieszania zakładamy, że wartości a i b są zmiennoprzecinkowe i należą do przedziału [0..1].

Dla obrazów 8-bitowych przy każdym mnożeniu tych wartości należałoby wynik podzielić przez 255, a w przypadku dzielenia wynik domnożyć przez 255. Na przykład niech $a = 255$ i $b = 255$. W wyniku mnożenia tych dwóch wartości $a \cdot b = 65025$ wynik ma wartość daleko większą niż dopuszczalna

wartość dla jednobajtowego obrazu i powinien być podzielony przez 255: $a \cdot b/255 = 255$. I analogicznie w wyniku dzielenia $a = 76$ przez $b = 153$ otrzymujemy wartość w przybliżeniu $a/b = 0.49$, którą należy pomnożyć przez wartość 255 dla przeskalowania wyniku na pełen 8-bitowy zakres: $a/b = 126$.

Tryby mieszania:

1. Suma (ang. *Additive mode*)
   Wartość wynikowa punktu jest klasyczną sumą dwóch składników:
   $$f(a, b) = a + b \tag{2.19}$$
   Przy czym wartość wyjściowa nie może przekroczyć wartości 1, zatem powinna to być suma z nasyceniem. Efekt działania trybu przedstawia Rysunek 2.9(a).
2. Odejmowanie (ang. *Subtractive mode*)
   $$f(a, b) = a + b - 1 \tag{2.20}$$
   Efekt działania trybu przedstawia Rysunek 2.9(b).
3. Różnica (ang. *Difference mode*)
   Wartością wyjściową jest wartość odejmowania od siebie składowych w takiej kolejności, żeby wynik był nieujemny:
   $$f(a, b) = |a - b| \tag{2.21}$$
   Efekt działania trybu przedstawia Rysunek 2.9(c).
4. Mnożenie (ang. *Multiply mode*)
   Mieszanie punktów w tym trybie jest klasycznym przemnożeniem wartości a oraz b:
   $$f(a, b) = a \cdot b; \tag{2.22}$$
   Globalnym efektem mnożenia jest przyciemnienie obrazu i zwiększenie jego kontrastu. Przy czym jasne punkty przyciemniane są bardziej. Efekt działania trybu przedstawia Rysunek 2.9(d).
5. Mnożenie odwrotności (ang. *Screen mode*)
   Rozjaśnianie polega na odwróceniu wartości punktów, przemnożeniu i ponownym odwróceniu uzyskanej wartości:
   $$f(a, b) = 1 - (1 - a) \cdot (1 - b) \tag{2.23}$$
   Efektem rozjaśniania będzie globalne zwiększenie jasności obrazu. Efekt działania trybu przedstawia Rysunek 2.9(e).
   Tryby Mnożenia i Mnożenia odwrotności są najpopularniejszymi technikami mieszania obrazów. W przypadku gdy $a = b$ oba tryby z powodzeniem zastępują klasyczne techniki zmiany jasności obrazu.

6. Negacja (ang. *Negation mode*)

$$f(a,b) = 1 - |1 - a - b| \tag{2.24}$$

Efekt działania trybu przedstawia Rysunek 2.9(f).

7. Ciemniejsze (ang. *Darken mode*)
Jeden z dwóch trybów wybierających wartości skrajne punktu, który wybiera zawsze ciemniejszą wartość.

$$f(a,b) = \begin{cases} a & \text{dla } a < b \\ b & \end{cases} \tag{2.25}$$

Efekt działania trybu przedstawia Rysunek 2.9(g).

8. Jaśniejsze (ang. *Lighten mode*)
Jeden z dwóch trybów wybierających wartości skrajne punktu, który wybiera zawsze jaśniejszą wartość.

$$f(a,b) = \begin{cases} a & \text{dla } a > b \\ b & \end{cases} \tag{2.26}$$

Efekt działania trybu przedstawia Rysunek 2.9(h).

9. Wyłączenie (ang. *Exclusion mode*)
Wyłączanie powoduje globalne obniżenie kontrastu obrazu.

$$f(a,b) = a + b - 2ab \tag{2.27}$$

Efekt działania trybu przedstawia Rysunek 2.9(i).

10. Nakładka (ang. *Overlay mode*)
Nakładka to połączenie trybu mnożenia i mnożenia odwrotności. Ciemniejsze punkty zostają przyciemnione a jaśniejsze rozjaśnione. Zastosowanie Nakładki na siebie daje efekt podobny do modyfikacji krzywą o kształcie S.

$$f(a,b) = \begin{cases} 2ab & \text{dla } a < 0.5 \\ 1 - 2 \cdot (1-a) \cdot (1-b) & \end{cases} \tag{2.28}$$

Efekt działania trybu przedstawia Rysunek 2.9(j).

11. Ostre światło (ang. *Hard light mode*)
Rozjaśnia punkty obrazu a, gdy kolor b jest jasny, a przy ciemnych wartościach b - przyciemnia kolory a. Wzmacnia kontrast w obrazie.

$$f(a,b) = \begin{cases} 2ab & \text{dla } b < 0.0 \\ 1 - 2 \cdot (1-a) \cdot (1-b) & \end{cases} \tag{2.29}$$

Efekt działania trybu przedstawia Rysunek 2.9(k).

12. Łagodne światło (ang. *Soft light mode*)
    Łagodniejsza wersja Ostrego światła, która nie zmienia globalnie kontrastu w obrazie.

$$f(a,b) = \begin{cases} 2ab + a^2 \cdot (1-2b) & \text{dla } b < 0.5 \\ \sqrt{a} \cdot (2b-1) + (2a) \cdot (1-b) \end{cases} \tag{2.30}$$

    Efekt działania trybu przedstawia Rysunek 2.9(l).
13. Rozcieńczenie (ang. *Color dodge mode*)
    Rozjaśnia punkty obrazu a ale tylko w tych miejscach, gdzie kolor b jest jasny.

$$f(a,b) = a/(1-b) \tag{2.31}$$

    Efekt działania trybu przedstawia Rysunek 2.9(m).
14. Wypalanie (ang. *Color burn mode*)
    Ciemna wartość punktu b przyciemnia kolor a poprzez zwiększenie kontrastu. Jasny kolor b powoduje nieznaczne zabarwienie koloru a.

$$f(a,b) = 1 - (1-a)/b \tag{2.32}$$

    Efekt działania trybu przedstawia Rysunek 2.9(n).
15. Reflect mode
    Zwiększa globalny kontrast obrazu.

$$f(a,b) = a^2/(1-b) \tag{2.33}$$

    Efekt działania trybu przedstawia Rysunek 2.9(o).
16. Przezroczystość (ang. *Transparency, Opacity*)
    Przezroczystość to stopień przenikania obrazu b przez obraz a. Jest to najprostszy i najczęściej używany sposób mieszania obrazów w całej grafice komputerowej zwany często z angielskiego "alfa blending". Stopień przenikania reguluje parametr $\alpha \in [0..1]$

$$f(a,b,\alpha) = (1-\alpha) \cdot b + \alpha \cdot a \tag{2.34}$$

    Dla obrazów 8-bitowych, mieszanie przezroczystości, z powodów wydajnościowych, definiuje się na liczbach całkowitych z przesunięciami bitowymi, a współczynnik przezroczystości α definiuje się w zakresie [0.225]:

$$f(a,b,\alpha) = (((255-\alpha) * b) >> 8) + ((\alpha * a) >> 8) \tag{2.35}$$

(a)

(b)

Rysunek 2.8. Dwa obrazy testujące tryby mieszania.

(a)

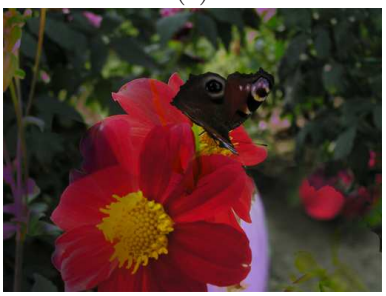
(b)

(c)

(d)

(e)

(f)



(g)

(h)

(i)

(j)

(k) (l)

(m) (n)

(o)

Rysunek 2.9. Tryby mieszania kolorów. (a) Suma (ang. *Additive mode*), (b) Odejmowanie (ang. *Subtractive mode*), (c) Różnica (ang. *Difference mode*), (d) Mnożenie (ang. *Multiply mode*), (e) Mnożenie odwrotności (ang. *Screen mode*), (f) Negacja (ang. *Negation mode*), (g) Ciemniejsze (ang. *Darken mode*), (h) Jaśniejsze (ang. *Lighten mode*), (i) Wyłączenie (ang. *Exclusion mode*), (j) Nakładka (ang. *Overlay mode*), (k) Ostre światło (ang. *Hard light mode*), (l) Łagodne światło (ang. *Soft light mode*), (m) Rozcieńczenie (ang. *Color dodge mode*), (n) Wypalanie (ang. *Color burn mode*), (o) Reflect mode

Rozdział 3

Transformacje geometryczne obrazów

Transformacja geometryczna obrazu zmienia wzajemne ułożenie punktów w obrazie. Takie przekształcenie składa się z dwóch podstawowych operacji: (1) przestrzennej transformacji współrzędnych oraz (2) interpolacji intensywności, która zostanie przypisana do przetransformowanych punktów. Transformacja współrzędnych może być wyrażona jako:

$$(x, y) = T\{(v, w)\} \tag{3.1}$$

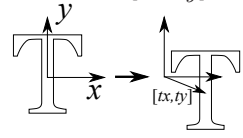
gdzie (v, w) są współrzędnymi punktu w oryginalnym obrazie, (x, y) są odpowiadającymi współrzędnymi w obrazie transformowanym, T jest operatorem transformacji. W ogólności jest olbrzymia ilość możliwych typów transformacji. W niniejszym rozdziale skupimy się na najbardziej podstawowym typie przekształcenia zwanym transformacją afiniczną i jego uproszczeniami.

3.1. Matematyczna notacja transformacji

Punkt na płaszczyźnie jest opisany dwoma współrzędnymi $[x, y]$ wyznaczającymi jego położenie względem początku układu współrzędnych. Rozważając układ kartezjański można zdefiniować kilka transformacji geometrycznych punktu. Najprostszą będzie przesunięcie punktu o określony wektor $[t_x, t_y]$ oraz obrócenie o pewien kąt φ względem początku układu współrzędnych. Ten zestaw przekształceń nazywa się ciałem sztywnym (ang. *rigid body*) , gdyż nie zmienia ani odległości pomiędzy punktami ani kątów pomiędzy prostymi. Do przekształcenia typu ciała sztywnego można dodać skalowanie (ang. *scale*) o współczynniki $[s_x, s_y]$ co w wyniku daje przekształcenie zwane RST , czyli przekształcenie niezmiennicze względem kątów, które może zmieniać odległości pomiędzy punktami ale nie może zmieniać kątów pomiędzy prostymi. Dodając pochylenia (ang. *shears*) i odbicia (ang. *reflections*) uzyskuje się przekształcenie afiniczne, czyli przekształcenie zachowujące jedynie równoległość linii. Przekształcenia te można zapisać w formie macierzowej we współrzędnych jednorodnych, uważając punkty z $\mathbb{R}^2$ za elementy przestrzeni $\mathbb{R}^3$ leżące w płaszczyźnie $z = 1$, gdzie $[x', y']$ są współrzędnymi punktu przed przekształceniem a $[x, y]$ współrzędnymi punktu po przekształceniu [53].

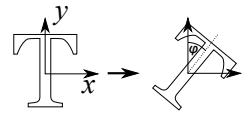
1. Translacja (przesunięcie) (ang. *shift, translation*) o wektor $[t_x, t_y]$:

$$[x, y, 1] = [x', y', 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$



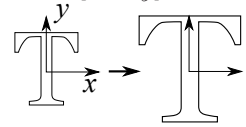
2. Rotacja (obrót) (ang. *rotation*) o kąt φ wokół początku układu współrzędnych:

$$[x, y, 1] = [x', y', 1] \begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



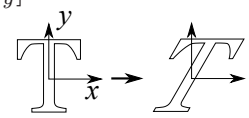
3. Skalowanie (ang. *scale*) o współczynnikach skalowania $[s_x, s_y]$:

$$[x, y, 1] = [x', y', 1] \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

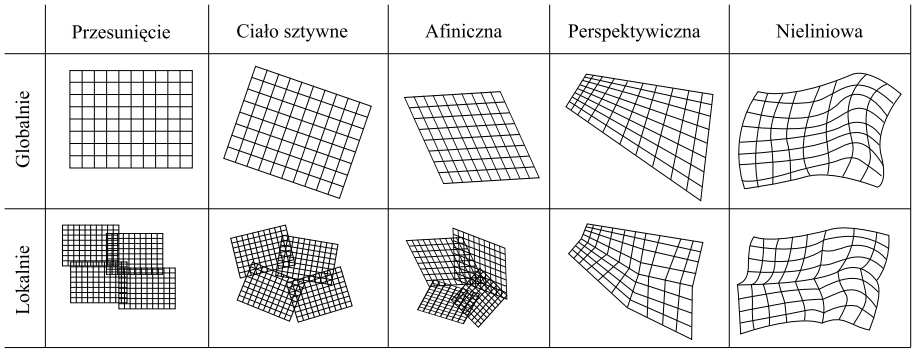


4. Pochylenie (ang. *shear*) o współczynnikach $[a_x, a_y]$:

$$[x, y, 1] = [x', y', 1] \begin{bmatrix} 1 & a_y & 0 \\ a_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Powyższe przypadki nie wyczerpują możliwych typów przekształceń ale z punktu widzenia przetwarzania obrazów są często wystarczające. Perspektywiczna transformacja jako jedyna dopuszcza każde inne liniowe przekształcenie ale liczba parametrów wzrasta do ośmiu. Tego typu przekształcenie zachowuje jedynie "prostość" linii. Nieliniowe metody, na przykład typu 'free-form deformation' lub elastyczne zazwyczaj rzutują linie proste w krzywe a ilość stopni swobody zależy głównie od stopnia krzywej. Rysunek 3.1 przedstawia obrazowo rodzaje transformacji dla przypadku przekształcenia globalnego i lokalnego.



Rysunek 3.1. Wizualne przedstawienie transformacji geometrycznych obrazu rozumianych globalnie i lokalnie.

W przypadku złożonych przekształceń można wykonywać sekwencyjnie każde przekształcenie z oddzielnych macierzy lub z jednej macierzy utworzonej w wyniku łączenia transformacji [28, 2]. Przykładowo, obrót punktu

o współrzędnych $[x, y]$ wokół punktu $[X_0, Y_0]$ można opisać zależnością:

$$[x, y, 1] = [x', y', 1][T * R * T^{-1}]$$

gdzie T oznacza macierz translacji o wektor $[-X_0, -Y_0]$, T^{-1} macierz do niej odwrotną a R macierz rotacji. W przypadku mnożenia macierzy kolejność ma znaczenie - mnożenie macierzy nie jest przemienne. Zatem dokonuje się następujących operacji: przesunięcia punktu obrotu $[X_0, Y_0]$ do początku układu współrzędnych, obrotu wokół początku układu współrzędnych, przesunięcia odwrotnego do wykonanego na początku.

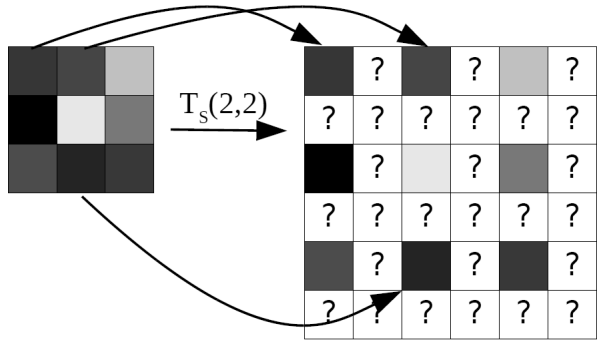
3.2. Implementacja transformacji afinicznej

Podczas procedury transformacji obrazu cyfrowego należy pamiętać, że bitmapa jest dwuwymiarową, dyskretną siatką i to współrzędne tej siatki są transformowane. Współrzędne siatki są zawsze liczbami całkowitymi, a kształt punktu jest w ogólności prostokątny. Jako zbiór punktów na płaszczyźnie każdy punkt rozważanego obrazu jest zatem przekształcany zgodnie z rozważanymi powyżej transformacjami. Będzie się to sprowadzało do przeliczenia współrzędnych każdego punktu zgodnie z macierzą transformacji na nowe położenie i przepisanie w nowe położenie wartości koloru. Tego typu podejście nazywa się przekształceniem w przód. Rozważmy jednak prosty przykład skalowania obrazu o współczynniki $[s_x = 2, s_y = 2]$ dające obraz wynikowy dwukrotnie powiększony w stosunku do oryginału. Niestety takie przekształcenie może powodować, że do niektórych punktów obrazu wynikowego może nie zostać przypisana żadna wartość koloru (tak jak na Rysunku 3.2) ewentualnie dwa lub więcej punktów oryginalnego obrazu może zostać zmapowanych na ten sam punkt obrazu wynikowego stwarzając problem połączenia tych wartości.

Aby uniknąć tego efektu stosuje się transformację odwrotną i przekształca się współrzędne punktów z obrazu wynikowego we współrzędne punktów obrazu źródłowego, czyli:

$$[x', y', 1] = [x, y, 1][M]^{-1}$$

Oczywiście przepisujemy wartości kolorów tych punktów, które po przekształceniu znalazły się w granicach źródłowego obrazu. Mając to na uwadze nowe współrzędne punktu, po przekształceniu trzeba przybliżyć do odpowiednich współrzędnych siatki definiującej obraz. Dla tego celu stosuje się szereg metod interpolacji, których dodatkowym zadaniem może być wygładzenie lub wyostrzenie obrazu.



Rysunek 3.2. Problem przekształcenia w przód przy skalowaniu obrazu.

3.3. Interpolacja obrazu

Interpolacja, czyli przybliżanie ma na celu określenie nowej wartości punktu na podstawie wartości punktu lub punktów sąsiadujących z rozważanym na obrazie oryginalnym. Sąsiedztwem punktu należy rozumieć punkty, które leżą bezpośrednio lub pośrednio przy danym punkcie. Rozważmy następujące przypadki interpolacji:

1. Interpolacja najbliższym sąsiadem
   Najprostsza metoda interpolacji, która używa intensywności punktu obrazu źródłowego leżącego w najbliższym węźle siatki. Współrzędne punktu po przekształceniu są zaokrąglane do najbliższej wartości całkowitej, a wartość nowego punktu obrazu docelowego będzie równa wartości punktu o zaokrąglonych współrzędnych z obrazu źródłowego. Ten typ interpolacji jest tani obliczeniowo i nie wymaga precyzji zmiennoprzecinkowej. Dodatkowo zachowuje intensywności punktów w obrazie, zatem histogram obrazu przed i po przekształceniu będzie podobny. Główną wadą tej metody jest powstawanie swego rodzaju "schodkowości" obrazu i braku gładkości, szczególnie widocznych przy krawędziach. Przykład zastosowania interpolacji najbliższym sąsiadem do powiększania obrazu jest przedstawiony na Rysunku 3.4(b).
2. Interpolacja biliniowa
   W przypadku interpolacji liniowej intensywność punktu I obliczana jest jako ważona suma 4 sąsiadów:

$$I(x) = \sum_i w_i I(x_i) \tag{3.2}$$

Wagi są obliczane z odległości pomiędzy pozycjami sąsiadów:

$$w_i = \prod_k (1 - |(x)_k - (x_i)_k|) \tag{3.3}$$

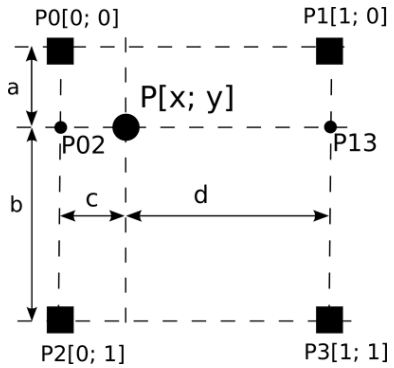
Przypuśćmy, że nowe współrzędne punktu $P = [x; y]$ w wyniku zadanego przekształcenia wypadły gdzieś pomiędzy czterema sąsiadami (zobacz Rysunek 3.3). Współrzędne te są liczbami zmiennoprzecinkowymi i wymagają dopasowania do punktów siatki. Pierwsza interpolacja, w pionie polega na uśrednieniu wartości punktów $P0$ z $P2$ i punktów $P1$ z $P3$ z wagami a i b. Wagi te są odległościami w pionie i w poziomie danego punktu od odpowiednich węzłów siatki. W wyniku tego uśrednienia dostajemy dwie nowe wartości:

$$P02 = a \cdot P2 + b \cdot P0 \tag{3.4}$$

$$P13 = a \cdot P3 + b \cdot P1 \tag{3.5}$$

Wartość wynikowego punku jest rezultatem drugiego uśrednianiem pomiędzy wartościami $P02$ i $P13$ z wagami c i d:

$$P = c \cdot P13 + d \cdot P03 \tag{3.6}$$



Rysunek 3.3. Schemat interpolacji biliniowej.

Pomimo identycznej złożoności obliczeniowej interpolacji najbliższym sąsiadem i biliniowej $O(n^2)$ interpolacja biliniowa jest kilkukrotnie wolniejsza od najbliższego sąsiada. Dodatkowo metoda wygładza krawędzie w obrazie, nieznacznie go rozmywa ale za cenę wprowadzenia nowych wartości punktów do obrazu co skutkuje zmianami w jego histogramie. Przykład zastosowania interpolacji biliniowej do powiększania obrazu jest przedstawiony na Rysunku 3.4(c).

3. Interpolacja kubiczna
   Interpolacja kubiczna rozszerza uśredniane sąsiedztwo do 16 punktów, a intensywność punktu liczona jest jako:

$$I(x) = \sum_{i=-1}^{2} I(u-i) f_i \tag{3.7}$$

   gdzie

$$f_{-1} = -\frac{1}{2}t^3 + t^2 - \frac{1}{2}t, \tag{3.8}$$

$$f_0 = \frac{3}{2}t^3 + \frac{5}{2}t^2 + 1, \tag{3.9}$$

$$f_1 = -\frac{3}{2}t^3 + 2t^2 + \frac{1}{2}t, \tag{3.10}$$

$$f_2 = \frac{1}{2}t^3 - \frac{1}{2}t^2 \tag{3.11}$$

   i $t = x' - u$ oraz u jest częścią całkowitą x. Pomimo wizualnego podobieństwa do interpolacji biliniowej bliższe badanie wykazuje duże różnice w wygładzaniu krawędzi i gradientów w obrazie, kosztem niestety kilkukrotnego spowolnienia całego procesu interpolacji. Przykład zastosowania interpolacji kublicznej do powiększania obrazu jest przedstawiony na Rysunku 3.4(d).
4. Interpolacja kubiczna b-sklejana (cubic spline)
   W interpolacji kubicznej b-sklejanej obraz jest reprezentowany przez funkcje b-sklejane rzędu czwartego. Intensywność punktu znajdującego się poza węzłem siatki w punkcie $0 \leq u \leq 1$ dla przypadku jednowymiarowego może być wyznaczona z

$$I_j = \sum_{i=-1}^{2} I_i' b_i(u) \tag{3.12}$$

   gdzie

$$b_{-1}(u) = (-u^3 + 3u^2 - 3u + 1)/6, \tag{3.13}$$

$$b_0(u) = (3u^3 - 6u^2 + 4)/6, \tag{3.14}$$

$$b_1(u) = (-3u^3 + 3u^2 + 3u + 1)/6, \tag{3.15}$$

$$b_2(u) = u^3/6 \tag{3.16}$$

   są krzywymi b-sklejanymi rzędu 4, a I_i są punktami kontrolnymi krzywej, czyli w rozważanym przypadku intensywnościami punktów z linii

obrazu źródłowego. W dwuwymiarowym przypadku intensywność punktu (u, v) wymaga użycia 16 punktów sąsiednich. Zatem obliczenie punktów kontrolnych powierzchni bikubicznej b-sklejanej w obrazie $n \times n$ wymaga n^2 równań, które pociągają za sobą n^4 mnożeń. Punkty kontrolne mogą być również uzyskane z filtracji odwrotnej. Przykład zastosowania interpolacji b-sklejanej do powiększania obrazu jest przedstawiony na Rysunku 3.4(e).

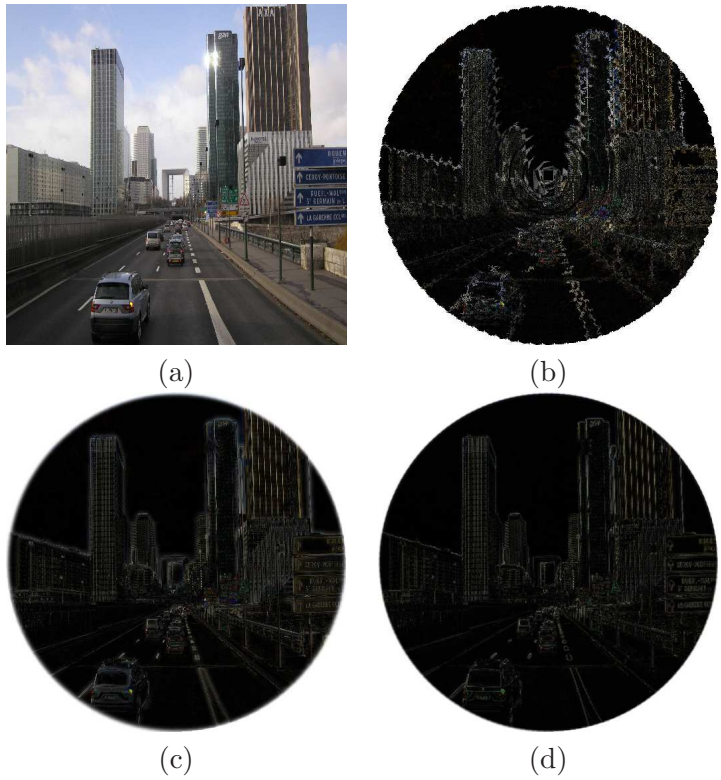
(a) (b) (c) (d) (e)

Rysunek 3.4. Interpolacja obrazu przy 8-krotnym powiększeniu: (a) obraz oryginalny w rozmiarze 64×64 oraz obrazy powiększone do rozmiaru 512×512 z zastosowaniem interpolacji: (b) najbliższym sąsiadem, (c) biliniowej, (d) kublicznej, (e) b-sklejanej.

Ten typ interpolacji można znacznie przyspieszyć wykorzystując Szybką Transformatę Fouriera, która obniża złożoność obliczeniową z $O(n^2)$ do $O(n \log n)$.

Złożoność obliczeniowa rozważanych metod interpolacji jest zebrana w

tabeli 3.1. Wpływ metod interpolacji na jakość transformacji wizualnie prezentuje Rysunek 3.5



(a) (b)

(c) (d)

Rysunek 3.5. Obraz oryginalny (a) został obrócony 90 razy o 4 stopnie za każdym razem z zastosowaniem interpolacji: (b) najbliższym sąsiadem, (c) liniowej, (d) kubicznej; wynikowy obraz został odjęty od obrazu oryginalnego. Wyraźnie widać wpływ stopnia interpolacji na jakości obrazu poddanemu transformacji.

Tabela 3.1. Złożoność obliczeniowa interpolacji przy przekształceniach obrazu dla przypadku najbliższego sąsiada, biliniowej, kubicznej, kubicznej typu spline. Przekształcany obraz w założeniu jest rozmiaru $n \times n$ punktów.

Typ interpolacji	Złożoność obliczeniowa
Najbliższy Sąsiad	$O(n^2)$
Biliniowa	$O(n^2)$
Kubiczna	$O(n^2)$
Kubiczna spline, liczona wprost	$O(n^4)$
Kubiczna spline, przy użyciu FFT	$O(n^2 \log n)$

3.4. Uwagi optymalizacyjne

Rozważane transformacje są dość procesorochłonne, zwłaszcza dla dużych 32-bitowych obrazów. Zakodowanie algorytmów wprost ze wzorów praktycznie nigdy nie jest dobrym pomysłem i skutkuje kiepskimi efektami wydajnościowe. W przypadku transformacji geometrycznych możliwych jest jednak szereg optymalizacji. Przede wszystkim operacje na liczbach całkowitych są wykonywane szybciej niż na liczbach zmiennoprzecinkowych, a dodawanie jest szybsze niż mnożenie, nie wspominając o dzieleniu. Oczywiście najszybsze będą operacje bitowe. Zatem warto, wymienione wyżej transformacje, zaimplementować na liczbach całkowitych z operacją dodawania (ewentualnie mnożenia) i przesunięć bitowych. Jeżeli potrzeba pewnych wartości z precyzją większą od liczby całkowitej można taką liczbę pomnożyć przez liczbę będącą potęgą dwójki (czyli przesunąć bitowo). Wykonać działania z tak powiększoną wartością a wynik podzielić przez tę potęgę dwójki (znowu przesunięcie bitowe). Przykładowo, jeżeli mamy mnożenie współrzędnych przez funkcje trygonometryczne, ze swej natury reprezentowane jako liczby zmiennoprzecinkowe, możemy przed pętlą wyznaczyć:

```
int isin = (int)sin(a)*256;
int icos = (int)cos(a)*256;
```

Przemnożenie przez 256 i rzutowanie na typ całkowity zaokrągli wartość zmiennoprzecinkową do około dwóch miejsc po przecinku i przesunie o 2 bity w lewo. Teraz, gdy potrzeba wartości sinusa i kosinusa użyć w pętli możemy zapisać:

```
1 int x = (( x*isin + y*icos )>>8);
```

Dzięki temu w pętli można pozbyć się działań na liczbach zmiennoprzecinkowych.

Drugim faktem godnym zauważenia jest sekwencyjność działań wykonywanych na obrazie. Pozwala to na wyeliminowanie mnożeń przez współrzędne x i y na rzecz dosumowywania przyrostów co obrót pętli. Dodatkowo, współczesne procesory wyposażone są w jednostki SIMD (ang. *Single Instruction, Multiple Data*) takie jak MMX czy SSE wykonujące daną operację na kilku zmiennych jednocześnie, co potrafi przyspieszyć operacje w pętli nawet kilkukrotnie. Niestety użycie tych jednostek wiąże się z niskopoziomowym programowaniem lub korzystaniem z dedykowanych bibliotek. Proszę pamiętać o wyliczaniu przed pętlą niezmienników. Bardzo dobre omówienie technik optymalizacyjnych i problemów związanych z geometrycznymi transformacjami obrazów w czasie rzeczywistym znajdzie czytelnik w pozycji [25].

Rozdział 4

Modele kolorów

Model kolorów to abstrakcyjny matematyczny sposób reprezentacji kolorów za pomocą liczb, najczęściej w postaci trzech lub czterech liczb – składowych koloru. Przestrzeń kolorów natomiast jest to model kolorów powiązany precyzyjnym opisem ze sposobem w jaki konkretne liczby opisujące kolor mają być interpretowane.

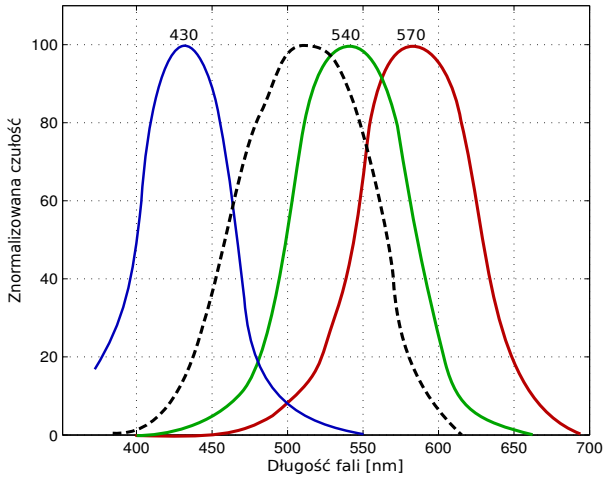
Dosyć naturalnym dla ludzkiego oka wydaje się rozbiór koloru na czerwień, zieleń i niebieski. Jest to mocno związane z fizjologią ludzkiego widzenia i budową oka. Światłoczułe receptory oka – czopki, w ilości około 5 milionów, skupione w środkowej części siatkówki, różnicują się w trzy rodzaje wrażliwości – na fale elektromagnetyczne o długościach $500 - 700nm$ interpretowanej jako barwa czerwona, o długościach $450-630nm$ interpretowanych jako barwa zielona oraz o długościach $400-500nm$ interpretowanych jako barwa niebieska. Widzenie fotopowe, za pomocą czopków daje obraz barwny ale tylko przy dostatecznej ilości światła padającego na siatkówkę oka, stąd też jest często nazywane widzeniem dziennym. Przy zmniejszającej się ilości światła widzenie fotopowe przechodzi przez widzenie mezopowe do widzenia skotopowego. Wyłączają wtedy swoje działanie czopki a za całość wrażeń wzrokowych odpowiada około 100 milionów pręcików ulokowanych na obrzeżach siatkówki. W odróżnieniu od czopków jest tylko jeden rodzaj pręcików zapewniając widzenie achromatyczne, pozbawione koloru. Pręciki mają również znacznie wyższą czułość w porównaniu do czopków, zatem zwiększa się czułość wzroku kosztem zmniejszenia ostrości widzenia [11]. Znormalizowaną wrażliwość oka na światło przy widzeniu fotopowym ilustruje Rysunek 4.1.

4.1. Model RGB

Model RGB jest podstawowym modelem przestrzeni barw stosowanym w technice cyfrowej. Nazwa modelu jest skrótem od pierwszych liter barw podstawowych w języku angielskim R–red (czerwony), B–green (zielony), B–blue (niebieski). Sam model może być wizualizowany jako trójwymiarowy sześcian odkładający każdą ze składowych na osiach X, Y, Z (zob. Rysunek 4.2).

Model RGB jest modelem addytywnym. W takiej przestrzeni brak składowych koloru (brak światła) oznacza kolor czarny, maksymalne wartości składowych dają w wyniku mieszania kolor biały. Identyczne wartości 3 składowych dają w rezultacie pewien stopień szarości, ciemniejszy lub jaśniejszy w zależności od intensywności składowych. Rysunek 4.3 przedstawia obraz RGB oraz jego komponenty – obrazy R, G oraz B.

RGB jest modelem zależnym od urządzenia (ang. *device-dependent*), tzn. nie jest w żaden sposób zobiektywizowany a różne urządzenia mogą do-



Rysunek 4.1. Znormalizowana czułość czopków oka ludzkiego w zależności od długości fali elektromagnetycznej. Linią przerywaną zaznaczona jest wrażliwość pręcików przy widzeniu skotopowym.

wolnie interpretować składowe koloru. Powoduje to szereg problemów jak choćby definicja czerni czy bieli. RGB nie definiuje również składowych kolorymetrycznie a ich mieszanie nie jest bezwzględne a zależne od składowych głównych. Stąd też potrzeba wspólnego zarządcy kolorów, np. ICC Color Managment [20]. Wtedy mówimy o przestrzeni kolorów, np. sRGB lub Adobe RGB. Szerzej w podrozdziale 4.4.

4.1.1. Skala szarości

Z trzech składowych R,G,B obrazu można uzyskać wartość punktu wyrażającą poziom szarości punktu licząc średnią arytmetyczną składowych:

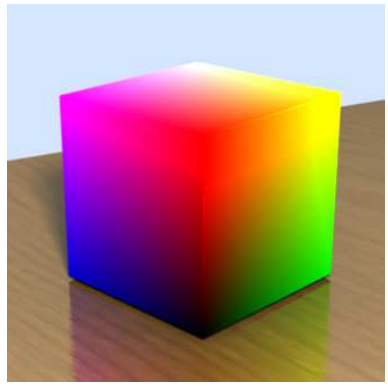
$$V = (R + G + B)/3 \tag{4.1}$$

Lepsze efekty, bliższe naturalnemu postrzeganiu człowiek, daje obliczenie średniej ważonej, np. wartości luminacji przestrzeni YUV (NTSC, PAL):

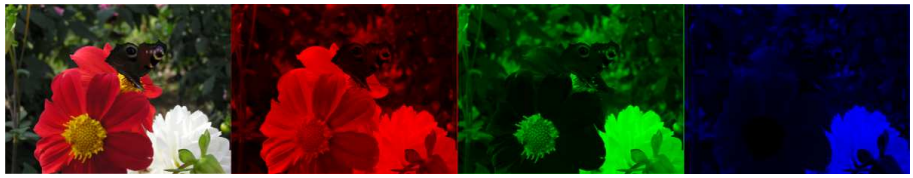
$$V = 0.299R + 0.587G + 0.114B \tag{4.2}$$

lub luminacji CIE (standard [ITU-BT709], HDTV]):

$$V = 0.2125R + 0.7154G + 0.0721B \tag{4.3}$$



Rysunek 4.2. Sześcian kolorów RGB.



Rysunek 4.3. Obraz RGB oraz rozseparowane obrazy składowych R, G i B

.

W obu przypadkach intensywność światła stara się naśladować krzywą wrażliwości pręcików oka ludzkiego (zobacz Rysunek 4.1). Wartość przybliżona dla 8 bitowych wartości, operując na liczbach całkowitych z przesunięciami bitowymi można wyrazić jako odpowiednio:

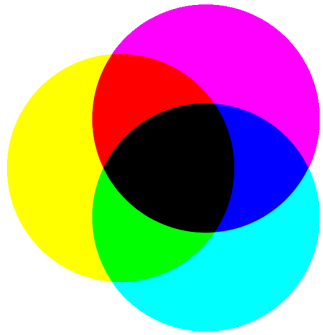
```
V = ( ( 77 * R + 150 * G + 29 * B) >> 8 )
V = ( ( 54 * R + 184 * G + 18 * B) >> 8 )
```

4.2. Model CMYK

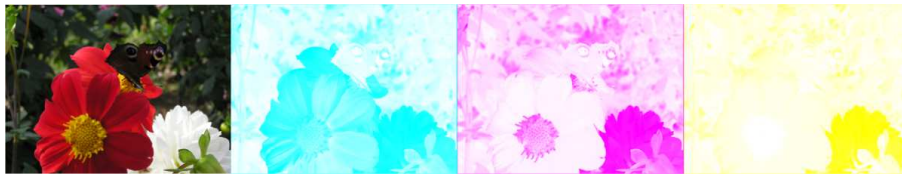
Model CMY powstał jako model subtraktywny do modelu RGB i jako taki posiada te same wady. Składowe modelu to C – cyjan (szafir, turkus), M – magenta (purpurowy, karmazynowy), Y – yellow (żółty). Rysunki 4.4 oraz 4.5 pokazują składanie barw modelu CMY. Model ten jest sam w sobie subtraktywny, tzn. brak składowych daje w wyniku kolor biały, natomiast wartości maksymalne składowych skutkują kolorem czarnym. Model ten zdecydowanie lepiej niż RGB nadaje się do reprodukcji barw w poligra-

fii gdzie medium stanowi na przykład biała kartka papieru a składowe są nanoszone w postaci farby drukarskiej.

Model CMY został rozszerzony o dodatkową składową K – blacK (czarny). Wynikało to z czysto praktycznych względów, bowiem trudno jest uzyskać głęboką czerń ze zmieszania podstawowych składowych, poza tym w przypadku druku byłoby to mocno nieopłacalne. Jednakże składowa czarna K nie jest składową niezależną od pozostałych a jej wartość jest wyznaczana jako część wspólna składowych CMY.



Rysunek 4.4. Subtraktywny model CMY.



Rysunek 4.5. Obraz RGB oraz rozseparowane obrazy składowych C, M i Y

.

Konwersja RBG na CMYK (Cyjan, Magenta, Yellow, blacK) dla wartości unormowanych $\in [0.0..1.0]$ wymaga najpierw wyznaczenia składowych modelu CMY:

$$[C_t, M_t, Y_t] = [1 - R, 1 - G, 1 - B]$$

Następnie wyznaczany jest poziom czerni jako najmniejszej z wartości C, M, Y a nowe wartości CMYK są przeliczane uwzględniając K jako część

wspólną koloru:

$$K = min(C_t, M_t, Y_t)$$
$$[C, M, Y, K] = \left[\frac{C_t - K}{1 - K}, \frac{M_t - K}{1 - K}, \frac{Y_t - K}{1 - K}, K\right]$$

Konwersja CMYK na RGB dla wartości unormowanych $\in$ [0.0..1.0] również wymaga przejściowego kroku:

$$[C_t, M_t, Y_t] = [C(1 - K) + K, M(1 - K) + K, Y(1 - K) + K]$$
$$[R, G, B] = [1 - C_t, 1 - M_t, 1 - Y_t]$$

4.3. Model HSL

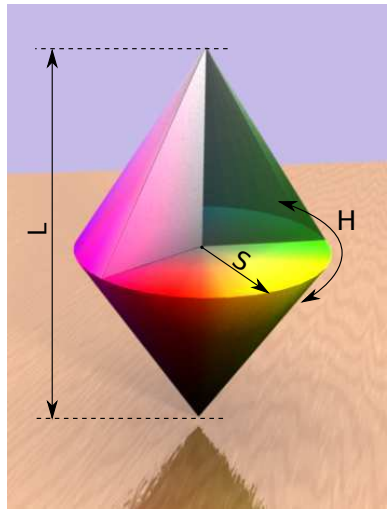
Jeden z cylindrycznych modeli barw zakładający, że postrzeganie barwy przez człowieka da się opisać przez trzy składowe L–jasność (ang. *Lightness*), S–nasycenie, saturacja (ang. *Saturation*) i H–barwa (ang. *Hue*) (innymi przykładami cylindrycznych modeli są HSV, HSI).

Składowa jasności L wyrażana procentowo w skali [0.0..1.0] lub [0..100] reprezentuje poziom światła białego składającego się na barwę i może być interpretowana jako natężenie wiązki światła. Składowa nasycenia S również jest wyrażana procentowo w skali [0.0..1.0] lub [0..100] i odpowiada za ilość "barwy w barwie". Przy wartości minimalnej S kolor jest stopniem szarości zależnym od składowej L, przy wartości maksymalnej uzyskany zostanie czysty kolor. Składowa ta może być interpretowana jako stopień monochromatyczności fali świetlnej. Barwa (składowa H) jest opisana liczbą z przedziału [0..360) i stanowi odpowiednik długości fali świetlnej, zatem to ta składowa decyduje o barwie koloru. Dla kąta $H = 0°$ otrzymujemy czystą czerwień, przez zieleń przy 120° i niebieski dla 270° aż do 360° zapętlając z powrotem do czerwieni.

Model HSL można opisać w postaci obustronnego stożka (zob. Rysunek 4.6). Wysokość stożka jest współrzędną L, odległość od środka stożka jest współrzędną S, kąt obrotu wokół wysokości stożka jest współrzędną H. Łatwo zauważyć, że współrzędne te nie są w pełni ortogonalne a niektóre kolory mogą być reprezentowane nieskończoną ilością współrzędnych. Przykładowo kolor szary z definicji ma saturację $S = 0$ a jego jasność zależy tylko od współrzędnej L, zatem barwa H może mieć w takim przypadku dowolną wartość.

Duża wadą modelu HSL jest brak możliwości opisania wszystkich barw. Model ten zakłada bowiem, że każdą barwę da się opisać za pomocą fali świetlnej o jednej tylko długości. Niestety oko ludzkie jest w stanie interpretować złożenie kilku fal świetlnych o różnej długości jako osobny kolor,

jak chociażby kolor purpurowy. Rysunek 4.7 obrazuje składowe modelu HSL interpretowane w skali szarości.



Rysunek 4.6. Schemat logicznego ułożenia bajtów RGB w 32-bitowym obrazie.

Rysunek 4.7. Obraz RGB oraz rozseparowane obrazy składowych kolejno L, S i H. Wartości składowych interpretowane jako skala szarości.

4.3.1. Konwersja modelu RGB do HSL

Składowe wejściowe R, G, B są unormowane do przedziału [0.0..1.0]. Składowa H $\in$ [0..360), składowe S i L $\in$ [0.0..1.0].

$$\begin{aligned} MAX &= \max(R, G, B) \\ MIN &= \min(R, G, B) \\ dM &= MAX - MIN; \end{aligned}$$

Składowa jasności L jest definiowana w następujący sposób:

$$L = \frac{MAX + MIN}{2}$$

W przypadku saturacji S należy rozpatrzeć trzy przypadki w zależności od wartości jasności L:

$$S = \begin{cases} 0 & \text{dla } L = 0 \text{ lub } MAX = MIN \\ \dfrac{dM}{2L} & \text{dla } 0 < L \leqslant 0.5 \\ \dfrac{dM}{2 - 2L} & \text{dla } L > 0.5 \end{cases}$$

W przypadku barwy H następuje najpierw sprawdzenie czy jest to kolor szary. Jeżeli tak to oznaczamy wartość H przez 0.

$$H = 0 \qquad \text{dla } MAX = MIN \tag{4.4}$$

W przeciwnym wypadku następuje zróżnicowanie w zależności od dominującej składowej:

$$H = \begin{cases} 60 * \dfrac{G - B}{dM} & \text{dla } MAX = R \text{ i } G \geqslant B \\ 60 * \dfrac{G - B}{dM} + 360 & \text{dla } MAX = R \text{ i } G < B \\ 60 * \dfrac{B - R}{dM} + 120 & \text{dla } MAX = G \\ 60 * \dfrac{R - G}{dM} + 240 & \text{dla } MAX = B \end{cases}$$

4.3.2. Konwersja modelu HSL do RGB

W konwersji odwrotnej najpierw następuje sprawdzenie wartości saturacji. Jeżeli jest równa 0 oznacza to kolor szary a składowe RGB są równe jasności:

$$R = G = B = L \qquad \text{dla } S = 0$$

Dla saturacji większej od 0 wyznaczanych jest kilka zmiennych pomocniczych:

$$Q = \begin{cases} L \cdot (1 + S) & \text{dla } L < 0.5 \\ L + S - (L \cdot S) & \text{dla } L \geqslant 0.5 \end{cases}$$

$$P = 2L - Q$$

$$H = H/360$$

$$T_r = H + 1/3$$

$$T_g = H$$

$$T_b = H - 1/3$$

$$\left.\begin{array}{l} T_c = T_c + 1 \text{ dla } T_c < 0 \\ T_c = T_c - 1 \text{ dla } T_c > 1 \end{array}\right\} \text{ dla każdego } T_c = [T_r, T_g, T_b]$$

Wynikową wartość każdej składowej RGB obliczamy z tego samego wyrażenia z odpowiednim $T_c = (T_r, T_g, T_b)$:

$$C = \begin{cases} P + (Q - P) \cdot 6 \cdot T_c & \text{dla } T_c < 1/6 \\ Q & \text{dla } 1/6 \leqslant T_c < 1/2 \\ P + (Q - P) \cdot 6 \cdot (2/3 - T_c) & \text{dla } 1/2 \leqslant T_c < 2/3 \\ P \end{cases}$$

4.4. Model CIE XYZ

CIE XYZ był jednym z pierwszych modeli kolorów zdefiniowanych matematycznie przez Międzynarodową Komisję Oświetleniową (ang. *International Commission on Illumination*, *fr. Comission Internationale de l'Eclairage – CIE*) w 1931 roku [8, 43]. Stanowi on swego rodzaju bazę dla następnych bardziej dopracowanych modeli opracowanych przez CIE. Jest to model trójchromatyczny, w którym CIE zdefiniowała układ trzech funkcji koloru $\overline{x}, \overline{y}, \overline{z}$, wywiedzione z funkcji czułości spektralnej oka standardowego obserwatora [18] (patrz Rysunek 4.8). Te funkcje są punktem wyjścia dla trzech teoretycznych składowych modelu XYZ:

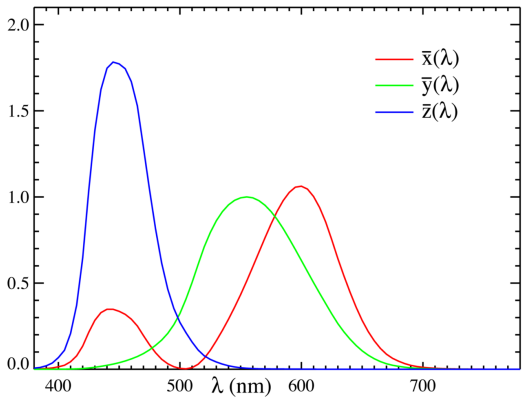
$$X = \int_0^\infty I(\lambda)\overline{x}(\lambda)d\lambda \tag{4.5}$$

$$Y = \int_0^\infty I(\lambda)\overline{y}(\lambda)d\lambda \tag{4.6}$$

$$Z = \int_0^\infty I(\lambda)\overline{z}(\lambda)d\lambda \tag{4.7}$$

Funkcja $I(\lambda)$ jest spektralnym rozkładem mocy światła padającego na siatkówkę oka standardowego obserwatora a λ jest długością równoważnej monochromatycznej fali świetlnej. Składowa Z z grubsza odpowiada wrażliwości na kolor niebieski, składowa Y odpowiada za luminację światła, natomiast składowa X jest liniową kombinacją wrażliwości czopków, dobraną w taki sposób żeby domykać całą przestrzeń kolorów i w dużej części pokrywa się z kolorem czerwonym.

Ze względów praktycznych wykres barw XYZ w odróżnieniu do modeli dyskutowanych w poprzednich podrozdziałach przedstawia się na tzw. wykresie chromatyczności. W tym celu CIE skonstruowała model CIE xyY



Rysunek 4.8. Trójchromatyczne składowe widmowe $\overline{x}, \overline{y}, \overline{z}$.

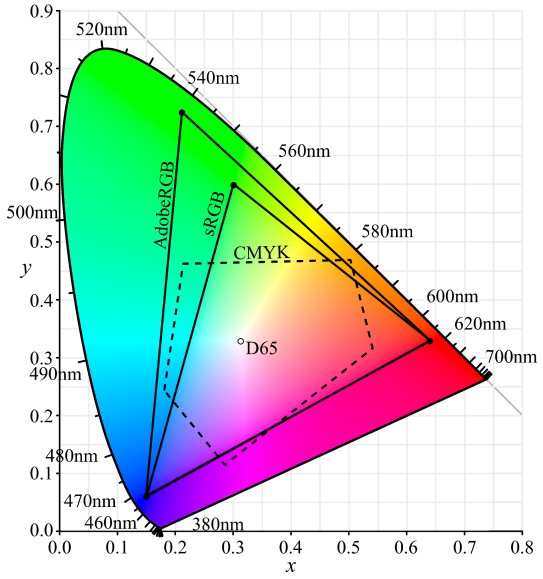
będący jedynie przekształceniem XYZ w ten sposób, że składowe x i y charakteryzują chromatyczność koloru a składowa Y jej jasność. Składowe x i y są wyrażone w następujący sposób:

$$x = \frac{X}{X+Y+Z} \tag{4.8}$$

$$y = \frac{Y}{X+Y+Z} \tag{4.9}$$

Rysunek 4.9 pokazuje wykres chromatyczności w modelu CIE xyY. W rzeczywistości jest to bryła przestrzenna a na wykresie przedstawiony jest jej przekrój dla ustalonej wartości współrzędnej Y. Kształt tej figury określa jedna prosta i jedna krzywa z określonymi długościami fali elektromagnetycznej podanymi w nanometrach.

Cała bryła modelu xyY obejmuje wszystkie barwy widzialne a związek składowych modelu z fizycznymi właściwościami fali świetlnej powoduje, że model CIEXYZ jest modelem niezależnym od urządzenia (ang. *device indepentent*). Zatem, aby przedstawić jakikolwiek kolor wybrany z palety XYZ za pomocą konkretnego urządzenia należy najpierw przekształcić go na model zależny od urządzenia. Najczęściej, w przypadku urządzeń emitujących promieniowanie (ekrany LCD, projektory, itp.) będzie to model RGB z odpowiednio zdefiniowaną podprzestrzenią. Dla poligrafii model XYZ będzie zwykle konwertowany do modelu CMYK. W obu modelach RGB i CMYK nie ma możliwości przedstawienia wszystkich możliwych widzialnych barw, zatem przy konwersji z modelu XYZ wycina się jedynie pewną bryłę zwaną przestrzenią modelu. Tak określony zakres możliwych do przedstawienia barw na danym urządzeniu nazywa się gamut (wymowa: gæmət) danej przestrzeni. Na Rysunku 4.9 zaznaczono gamut w postaci trójkąta dla naj-



Rysunek 4.9. Wykres chromatyczności modelu CIE xyY. Mniejszy trójkąt określa gamut przestrzeni sRGB, większy określa gamut przestrzeni AdobeRGB, przerywaną linią zaznaczono typowy gamut przestrzeni CMYK. Małym okręgiem zaznaczono referencyjny punkt bieli D65.

popularniejszych przestrzeni sRGB i AdobeRGB oraz typowej przestrzeni CMYK.

Przestrzeń sRGB (standarised Red, Green, Blue) [20] została zaproponowana wspólnie przez Microsoft i Hewlett-Packard w 1996 roku do zastosowań nieprofesjonalnych w tym jako standard internetowy. W przestrzeni tej można odwzorować po 256 wartości dla każdej z trzech składowych co daje w wyniku mieszania nieco ponad 16 milionów możliwych kolorów. Jest to przestrzeń, w której gamut jest stosunkowo nieduży, bo pokrywa około 35% wszystkich widzialnych barw ale w zamian większość popularnych urządzeń jest w stanie prawidłowo ją odwzorować.

Do zastosowań bardziej profesjonalnych firma Adobe w 1998 roku zdefiniowała przestrzeń AdobeRGB [1]. Ta przestrzeń obejmuje ok. 50% wszystkich widzialnych barw, zwiększając gamut w stosunku do sRGB głównie w zieleniach i cyjanie kosztem większych wymagań od urządzeń obrazujących. Istnieje oczywiście o wiele więcej standardów przestrzeni kolorów, które jednak wykraczają poza ramy niniejszej pozycji.

Przy konwersji pojawia się problem zdefiniowania referencyjnego punktu odniesienia, za który przyjmuje się kolor biały (ang. *reference white point*). Dla różnych przestrzeni kolor biały może mieć różne współrzędne na wykre-

sie chromatyczności. Co więcej kolor biały będzie się różnił w zależności od obserwatora czy pory dnia lub typu sztucznego oświetlenia. Przykładowo, w Europie, za referencyjny punkt bieli światła dziennego (ang. *daylight reference white point*) przyjmuje się kolor ciała doskonale czarnego o temperaturze 6500K. Punkt ten ma oznaczenie D65 i w modelu CIE XYZ znajduje się na współrzędnych $X_r = 0.9505, Y_r = 1.0000, Z_r = 1.0891$.

Sama konwersja modelu CIE XYZ do RGB wymaga określenia macierzy konwersji zdefiniowanej dla konkretnej przestrzeni RGB. W następnych podrozdziałach zostanie przedstawiona konwersja dla przypadku przestrzeni sRGB oraz AdobeRGB.

4.4.1. Konwersja modelu RGB do CIE XYZ

Składowe R, G, B są unormowane do [0.0..1.0] i podniesione do potęgi Gamma=2.2: $C = c^{\gamma}$ [14]

$$[X, Y, Z] = [R, G, B][M]$$

Macierz M wynosi dla przestrzeni Adobe RGB (1998):

$$\mathrm{M}= \begin{vmatrix} 0.57667 & 0.29734 & 0.02703 \\ 0.18556 & 0.62736 & 0.07069 \\ 0.18823 & 0.07529 & 0.99134 \end{vmatrix}$$

Dla przestrzeni sRGB:

$$\mathrm{M}= \begin{vmatrix} 0.41242 & 0.21266 & 0.01933 \\ 0.35759 & 0.71517 & 0.11919 \\ 0.18046 & 0.07218 & 0.95044 \end{vmatrix}$$

Obie przestrzenie odnoszą się do referencyjnego punktu bieli D65 (temperatura bieli = 6500K) o składowych: $X_r = 0.9505, Y_r = 1.0000, Z_r = 1.0891$.

4.4.2. Konwersja modelu CIE XYZ do RGB

Składowe R, G, B są unormowane do [0.0..1.0] i podniesione do potęgi $C = c^{\frac{1}{\gamma}}$, gdzie Gamma=2.2

$$[R, G, B] = [X, Y, Z][M]$$

Macierz M wynosi dla przestrzeni Adobe RGB (1998):

$$\mathrm{M}= \begin{vmatrix} 2.04159 & -0.96924 & 0.01344 \\ -0.56501 & 1.87597 & -0.11836 \\ -0.34473 & 0.04156 & 1.01517 \end{vmatrix}$$

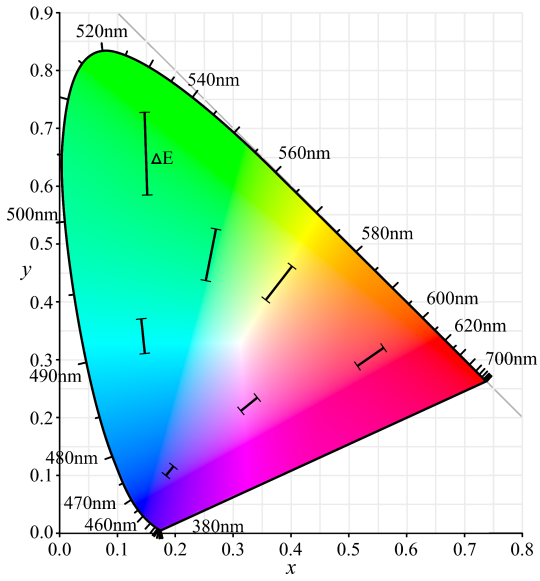
Dla przestrzeni sRGB:

$$M = \begin{vmatrix} 3.24071 & -0.96925 & 0.05563 \\ -1.53726 & 1.87599 & -0.20399 \\ -0.49857 & 0.04155 & 1.05707 \end{vmatrix}$$

Obie przestrzenie odnoszą się do referencyjnego punktu bieli D65 (temperatura bieli = 6500K) o składowych: $X_r = 0.9505, Y_r = 1.0000, Z_r = 1.0891$.

4.5. Modele CIE L*a*b* oraz CIE L*u*v*

Model CIE XYZ opracowany w 1931 roku mimo swoich niewątpliwych zalet miał też wady. Najpoważniejszą z nich był brak percepcyjnej równomierności (ang. *perceptual uniformity*), czyli niejednakowa odległość euklidesowa w przestrzeni kolorów pomiędzy identycznymi różnicami kolorów. Okazało się bowiem, że standardowy obserwator w okręgu o jednakowym promieniu umieszczonym na wykresie chromatyczności w obszarze kolorów niebieskich jest w stanie wyróżnić więcej przejść tonalnych niż w okręgu o identycznym promieniu umieszczonym w obszarze kolorów zielonych. Inaczej mówiąc odległość w modelu XYZ, na której obserwator jest w stanie wyróżnić jednakową ilość przejść tonalnych, oznaczaną przez ΔE jest zależna od badanego koloru. Rysunek 4.10 obrazuje odległość ΔE dla kilku wybranych kolorów na wykresie chromatyczności.



Rysunek 4.10. Percepcyjna nierównomierność modelu XYZ.

Prace nad równomiernością modelu CIE XYZ doprowadziły w 1976 roku do powstania nowych modeli barw będących matematycznym przekształce-

niem modelu CIE XYZ: CIE L*a*b* oraz CIE L*u*v*. W obu przypadkach oprócz wprowadzenia równomierności percepcyjnej dokonano separacji luminacji (kanał L) od kanałów chrominacji.

Model CIE L*a*b* jest matematyczną nieliniową transformacja modelu CIE XYZ [13, 43]. Barwa w tym modelu jest opisywana za pomocą trzech składowych: L^* – luminacja $\in$ [0..100], własność achromatyczna od czarnego do białego, a^* – składowa chromatyczna od zielonej do magenty $\in$ [−128..127], b^* – składowa chromatyczna od niebieskiej do żółtej $\in$ [−128..127]. Składowe chromatyczne modelu są zdefiniowane jako barwy przeciwstawne: kolor zimny – kolor ciepły. Barwy tak skonstruowane wykluczają się wzajemnie, tzn. kolor nie może być równocześnie niebieski i żółty czy czerwony i zielony.

Co więcej w modelu CIE L*a*b* da się tak dobrać współrzędne L, a, b, że barwa opisana takimi wartościami liczbowymi nie istnieje w świecie rzeczywistym. Zwykle są to współrzędne $a^*, b^* > |120|$. Zatem model ten zawiera część kolorów nie mających swojego fizycznego odpowiednika. Stwarza to oczywiście pewne problemy przy przetwarzaniu obrazów z wykorzystaniem tego modelu. Pomimo istnienia takich wirtualnych kolorów model ten jest bardzo popularny i stanowi podstawę dla systemów zarządzania barwą, jak chociażby profile ICC [20]. Trójwymiarową wizualizację gamut modelu CIE L*a*b* czytelnik znajdzie na stronie Bruce'a Lindblooma [24]

Model CIE L*u*v* jest również matematyczną nieliniową transformacja modelu CIE XYZ [43]. Prostszy obliczeniowo sposób transformacji modelu CIE XYZ sprawia, że jest to model chętnie wykorzystywany w grafice komputerowej gdy zachodzi potrzeba dużej optymalizacji obliczeniowej. Analogicznie do modelu Lab składowa L^* określa jasność (luminację) barwy i mieści się w zakresie $< 0.0; 100.0 >$. Współrzędne u^* i v^* są składowymi chromatycznymi opisującymi odpowiednio zakres barw od zieleni do czerwieni oraz od niebieskiego do żółtego. Składowa u^* ma całkowity zakres [−134..220] a składowa v^* – [−140..122]. Analogicznie jak to było w przypadku modelu Lab tak i tu istnieją nierzeczywiste kolory głównie w zakresach $u*, v* > |120|$.

Odległość euklidesowa pomiędzy barwami ΔE w modelu CIE L*a*b* jest zdefiniowana jako:

$$\Delta E = \sqrt{(\Delta L)^2 + (\Delta a)^2 + (\Delta b)^2} \tag{4.10}$$

i analogicznie dla CIE L*u*v*:

$$\Delta E = \sqrt{(\Delta L)^2 + (\Delta u)^2 + (\Delta v)^2} \tag{4.11}$$

Standardowy obserwator jest w stanie odróżnić barwy od siebie gdy ΔE jest większa od około 2.

Aby opisać model definiuje się dodatkowo pojęcia chromy (ang. *chromacity*) i odcienia (ang. *hue*) danego koloru. Chroma jest euklidesową odległością danej barwy (L, a, b) od referencyjnego punktu bieli (L, a_r, b_r) na wykresie chromatyczności opisaną przez wyrażenie:

$$c = 13L\sqrt{(a - a_r)^2 + (b - b_r)^2} \tag{4.12}$$

a odcień jest jej kątem:

$$h = arctg\frac{(a - a_r)}{(b - b_r)} \tag{4.13}$$

Analogicznie dla modelu CIE L*u*v* zastępując składowe a i b przez u i v.

4.5.1. Konwersja modelu CIE XYZ do CIE L*a*b*

Model CIE L*a*b* jest matematyczną nieliniową transformacja modelu CIE XYZ, zatem żeby przekształcić wartość koloru modelu Lab na odpowiedni kolor w modelu RGB niezbędny jest krok pośredni, czyli konwersja na model CIE XYZ.

$$L = 116f_y - 16 \tag{4.14}$$

$$a = 500(f_x - f_y) \tag{4.15}$$

$$b = 200(f_y - f_z) \tag{4.16}$$

Gdzie:

$$f_x = \begin{cases} \sqrt[3]{x_r} & x_r > \epsilon \\ \dfrac{\kappa x_r + 16}{116} & x_r \leqslant \epsilon \end{cases} \tag{4.17}$$

$$f_y = \begin{cases} \sqrt[3]{y_r} & y_r > \epsilon \\ \dfrac{\kappa y_r + 16}{116} & y_r \leqslant \epsilon \end{cases} \tag{4.18}$$

$$f_z = \begin{cases} \sqrt[3]{z_r} & z_r > \epsilon \\ \dfrac{\kappa z_r + 16}{116} & z_r \leqslant \epsilon \end{cases} \tag{4.19}$$

$$x_r = \frac{X}{X_r} \tag{4.20}$$

$$y_r = \frac{Y}{Y_r} \tag{4.21}$$

$$z_r = \frac{Z}{Z_r} \tag{4.22}$$

Stała $\epsilon = 0.008856$, stała $\kappa = 903.3$. Współrzędne X_r, Y_r, Z_r są składowymi modelu CIE XYZ odnoszącymi się do referencyjnego poziomu bieli.

4.5.2. Konwersja modelu CIE L*a*b* do CIE XYZ

Konwersja odwrotna również wymaga pośredniego modelu CIE XYZ.

$$X = x_r X_r \tag{4.23}$$
$$Y = y_r Y_r \tag{4.24}$$
$$Z = z_r Z_r \tag{4.25}$$

Gdzie:

$$x_r = \begin{cases} f_x^3 & f_x^3 > \epsilon \\ \dfrac{116 f_x - 16}{\kappa} & f_x^3 \leqslant \epsilon \end{cases} \tag{4.26}$$

$$y_r = \begin{cases} \left(\dfrac{L+16}{116}\right)^3 & L > \kappa\epsilon \\ \dfrac{L}{\kappa} & L \leqslant \kappa\epsilon \end{cases} \tag{4.27}$$

$$z_r = \begin{cases} f_z^3 & f_z^3 > \epsilon \\ \dfrac{116 f_z - 16}{\kappa} & f_z^3 \leqslant \epsilon \end{cases} \tag{4.28}$$

$$f_x = \frac{a}{500} + f_y \tag{4.29}$$
$$f_z = f_y - \frac{b}{200} \tag{4.30}$$
$$f_y = \begin{cases} \dfrac{L+16}{116} & y_r > \epsilon \\ \dfrac{\kappa y_r + 16}{116} & y_r \leqslant \epsilon \end{cases} \tag{4.31}$$

Stała $\epsilon = 0.008856$, stała $\kappa = 903.3$. Współrzędne X_r, Y_r, Z_r są składowymi XYZ odnoszącymi się do referencyjnego poziomu bieli.

4.5.3. Konwersja modelu CIE XYZ do CIE L*u*v

Matematyczna transformacja przestrzeni CIE XYZ do CIE L*u*v*, $L^* \in [0..100]$, $u^* \in [-134..220], v^* \in [-140..122]$.

$$L^* = \begin{cases} 116\sqrt[3]{\frac{Y}{Y_n}} - 16 & \frac{Y}{Y_n} > \epsilon \\ \kappa\left(\frac{Y}{Y_n}\right) & \left(\frac{Y}{Y_n}\right) \leqslant \epsilon \end{cases} \tag{4.32}$$

$$u^* = 13L(u' - u'_n) \tag{4.33}$$

$$v^* = 13L(v' - v'_n) \tag{4.34}$$

gdzie:

$$u' = \frac{4X}{(X + 15Y + 3Z)} \tag{4.35}$$

$$v' = \frac{9Y}{(X + 15Y + 3Z)} \tag{4.36}$$

Wartości Y_n, u'_n, v'_n odnoszą się do referencyjnego punktu bieli. Dla typowej sytuacji przyjmuje się: $Y_n = 1.0, u'_n = 0.2009, v'_n = 0.4610$. Stała $\epsilon = 0.008856$, stała $\kappa = 903.3$.

4.5.4. Konwersja modelu CIE L*u*v do CIE XYZ

Konwersja odwrotna jest zdefiniowana w następujący sposób:

$$Y = \begin{cases} Y_n \dfrac{L^*}{\kappa} & L^* \leqslant \kappa\epsilon \\ Y_n \left(\dfrac{L^* + 16}{116}\right)^3 & L^* > \kappa\epsilon \end{cases} \tag{4.37}$$

$$X = Y\frac{9u'}{4v'} \tag{4.38}$$

$$Z = Y\frac{12 - 3u' - 20v'}{4v'} \tag{4.39}$$

gdzie:

$$u' = \frac{u}{13L^*} + u'_n$$

$$v' = \frac{v}{13L^*} + v'_n$$

Wartości Y_n, u'_n, v'_n odnoszą się do referencyjnego punktu bieli. Dla typowej sytuacji przyjmuje się: $Y_n = 1.0, u'_n = 0.2009, v'_n = 0.4610$. Stała $\epsilon = 0.008856$, stała $\kappa = 903.3$.

Rozdział 5

Filtracja cyfrowa obrazów

Filtry cyfrowe działające w przestrzeni obrazu (ang. *spatial filtering*) to jedno z podstawowych narzędzi używanych w przetwarzaniu obrazów. Sama nazwa "filtry" została zapożyczona z przetwarzania w dziedzinie częstotliwości (ang. *frequency filtering*), które jest omawiane w rozdziale 7, gdzie "filtracja" odnosi się do przepuszczania bądź tłumienia pewnych, określonych częstotliwości sygnału cyfrowego. I rzeczywiście filtracja w dziedzinie częstotliwości będzie miała te same możliwości, które daje filtracja splotowa, jednakże filtracja przestrzenna jest znacznie bardziej uniwersalna, jak chociażby w przypadku filtracji nieliniowej.

W poniższym rozdziale mianem filtrów cyfrowych będą określane wszelkie operacje, które można zapisać jako:

$$g(x,y) = T[f(x,y)] \tag{5.1}$$

gdzie $f(x,y)$ jest obrazem wejściowym, $g(x,y)$ jest obrazem wyjściowym a T jest operatorem działającym nad f w pewnym otoczeniu (sąsiedztwie, ang. *neighborhood*) punktu o współrzędnych (x,y). Zwykle sąsiedztwo jest definiowane jako prostokątny obszar, z punktem centralnym we współrzędnych (x,y) zdecydowanie mniejszy niż wielkość obrazu. Nie jest to jednak ścisła definicja a sąsiedztwo w ogólności może być definiowane dowolnie. Najmniejsze możliwe do zdefiniowania sąsiedztwo na obszarze 1×1, w którym wynikowy punkt $g(x,y)$ zależy tylko od odpowiadającego mu punktu $f(x,y)$ i operatora T prowadzi do transformacji intensywności obrazu omawianej w rozdziale 2. Sama operacja T działająca w sąsiedztwie punktu jest nazywana filtrem przestrzennym lub maską filtru (ewentualnie oknem filtru lub zapożyczonym z języka angielskiego kernelem filtru).

Zastosowanie filtracji cyfrowej w przestrzeni obrazu daje olbrzymie możliwości modyfikacji obrazu. Filtry rozmywające, uśredniające mają właściwości wygładzające i redukujące szum, filtry krawędziowe i wyostrzające wydobywają z obrazu niedostrzegalne szczegóły a dzięki filtrom nieliniowym możliwa staje się rekonstrukcja częściowo uszkodzonego obrazu. Rozdział ten zacznie się od omówienia najbardziej uniwersalnego filtru jakim jest filtr splotowy (ang. *convolution filter*) a następnie zostaną omówione filtry nieliniowe i statystyczne.

5.1. Filtracja splotowa

Splot (konwolucja, ang. *convolution*) jest matematyczną operacją łączenia (splatania) dwóch sygnałów w celu utworzenia trzeciego. Można zaryzykować stwierdzenie, że jest to jedna z najważniejszych operacji w cyfrowym przetwarzaniu sygnałów. Splot dwóch funkcji f i g jednej zmiennej, ciągłych i całkowalnych jest zdefiniowany jako całka z mnożenia dwóch funkcji, przy

czym jedna z nich jest odwrócona i przesuwana. Rozważając funkcje f i g można zdefiniować ich splot jako:

$$f(x) \otimes g(x) \stackrel{def}{=} \int_{-\infty}^{\infty} f(t)g(x-t)dt = \varphi(x) \tag{5.2}$$

Graficzna interpretacja operacji splotu jest pokazana na Rysunku 5.1.
Tak zdefiniowana operacja ma szereg istotnych własności algebraicznych:

— przemienność

$$f \otimes g = g \otimes f \tag{5.3}$$

— łączność

$$(f \otimes g) \otimes h = f \otimes (g \otimes h) \tag{5.4}$$

— rozdzielność względem dodawania

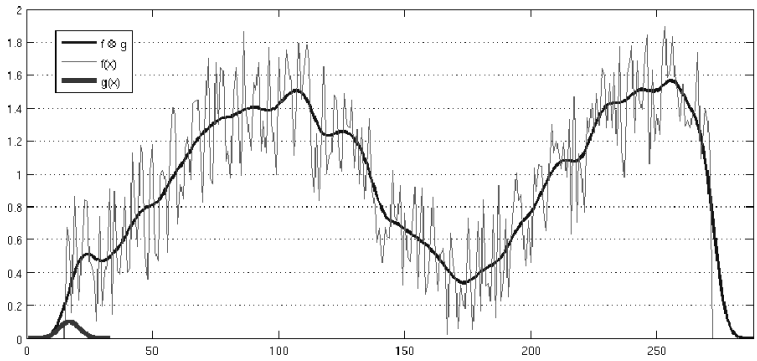
$$f \otimes (g + h) = f \otimes g + f \otimes h \tag{5.5}$$

— łączność względem mnożenia przez skalar

$$\alpha(f \otimes g) = (\alpha f) \otimes g = f \otimes (\alpha g) \tag{5.6}$$

Dodatkowo, przy zastosowaniu operacji splotu do filtracji sygnałów istotna jest jeszcze, tzw. impulsowa odpowiedź filtru. Daje ona informację jak zachowuje się filtr gdy na jego wejściu pojawi się funkcja Diraca δ. Odpowiedzią filtru będzie:

$$h(x) \otimes \delta(x) = \int_{-\infty}^{\infty} h(t)\delta(x-t)dt = h(x) \tag{5.7}$$



Rysunek 5.1. Ilustracja operacji splotu dwóch funkcji $f(x)$ i $g(x)$. Mocno zaszumiony sygnał f i sygnał z nim splatany g daje w wyniku trzeci sygnał - wynik filtracji.

W przetwarzaniu obrazów taką odpowiedź impulsową nazywa się zazwyczaj maską filtru, maską splotu, albo po prostu filtrem. Obraz jako sygnał dyskretny i dwuwymiarowy potrzebuje zdefiniowania operacji splotu w wersji dyskretnej. Podsumowując, splot filtru $h(x, y)$ o rozmiarze $S \times T$ z funkcją intensywności obrazu $I(x, y)$ o rozmiarze $M \times N$ jest wyrażony równaniem:

$$I(x,y) \otimes h(x,y) = \sum_{i=-s}^{s} \sum_{j=-t}^{t} h(i,j) I(x+i, y+j) \tag{5.8}$$

gdzie $s = (S-1)/2$, $t = (T-1)/2$, przy założeniu, że S i T są nieparzystymi liczbami całkowitymi. Dla określania wielkości maski filtru często używa się promienia filtru zdefiniowanego jako wektor $r = [s, t]$. Wtedy, w najczęstszym przypadku maski kwadratowej jej wielkość można opisać poprzez promień jako $S = T = 2r + 1$.

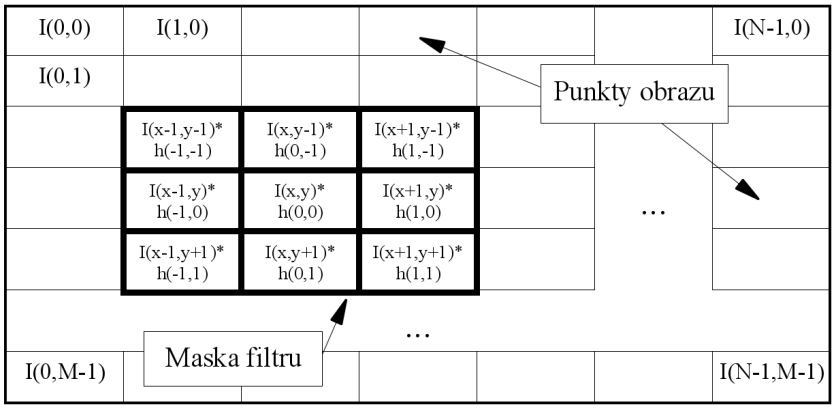
Z wyrażenia (5.8) można wywnioskować, że na każdy punkt obrazu wynikowego ma wpływ grupa punktów z obrazu źródłowego z jego otoczenia. O tym z jaką siłą oddziałują poszczególne punkty na wartość punktu wynikowego decyduje maska filtru a dokładniej wartości jej składowych elementów. Z reguły, wartości maski filtru zbiera się w tablicę dwuwymiarową złożoną ze współczynników $h(i, j)$. Z powodów wydajnościowych elementy maski są liczbami całkowitymi a dodatkowo wielkość maski w poziomie i w pionie jest liczbą nieparzystą co zapewnia istnienie elementu środkowego. Stosowanie liczb całkowitych powoduje, że zachodzi potrzeba unormowania jasności punktu wynikowego, zmieniając wyrażenie (5.8) na:

$$I(x,y) \otimes h(x,y) = \frac{\sum_{i=-s}^{s} \sum_{j=-t}^{t} h(i,j) I(x+i, y+j)}{\sum_{i=-s}^{s} \sum_{j=-t}^{t} h(i,j)} \tag{5.9}$$

To wyrażenie jest obliczane dla wszystkich punktów (x, y) obrazu I, dla których otoczenie w danym punkcie jest zdefiniowane. Wyrażenie w mianowniku jest często nazywane **wagą maski** filtru. Rysunek 5.2 ilustruje operację splotu filtru h z obrazem I.

Sama numeryczna operacja splotu będzie się sprowadzała do iteracji po każdym punkcie obrazu I i zsumowania iloczynów wartości punktów obrazu z jego otoczenia z wartościami maski h. Uzyskaną wartość należy jeszcze podzielić przez wagę maski dla przeskalowania wyniku tak aby jego zakres mieścił się w 8-bitowym zakresie. Z uwagi na to, że w składowych maski mogą pojawić się wartości ujemne waga maski nie zawsze będzie wartością dodatnią. W przypadku gdy suma wag maski jest równa 0 za wagę maski można przyjąć wartość równą 1. W przypadku gdy wartość

splotu w danym punkcie osiąga wartość ujemną, w zależności od rodzaju filtru, można: (1) przyciąć wartość ujemną do zera, (2) wyznaczyć wartość bezwzględną splotu, (3) przeskalować wartości wynikowego obrazu tak żeby najmniejsza wartość splotu miała wartość 0 a największa 255.

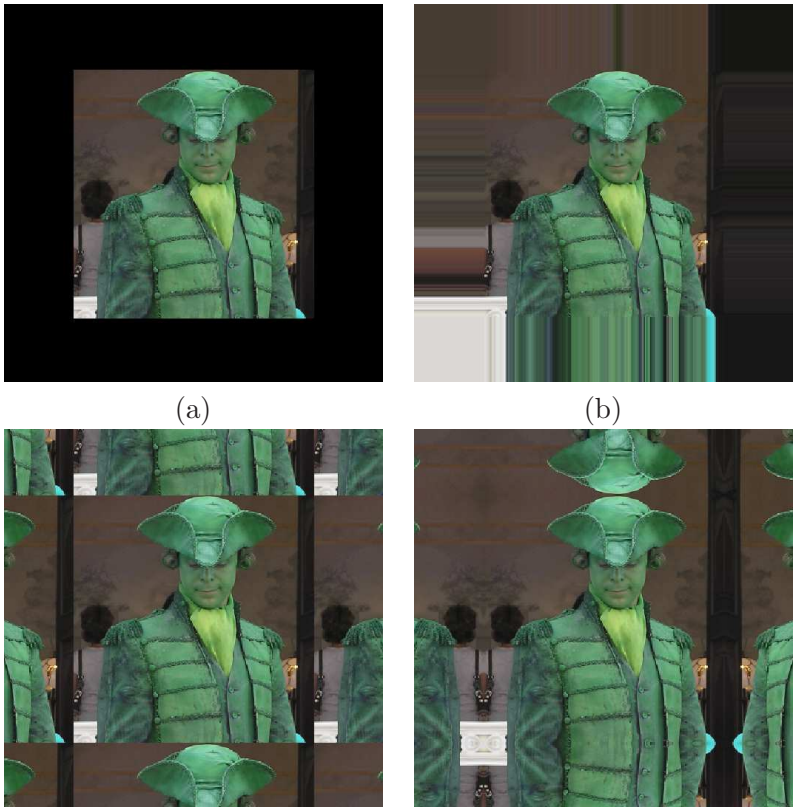


Rysunek 5.2. Mechanizm filtracji splotowej obrazu I przy użyciu maski h o rozmiarze 3×3. Przemnożone przez współczynniki maski wartości jasności obrazu zostaną zsumowane i podzielone przez sumę współczynników maski.

Jak w każdym przypadku operacji splotu także i tu pojawia się klasyczny problem brzegowy, czyli problem obliczenia splotu na krawędzi sygnału, gdzie jego otoczenie jest niezdefiniowane. W najprostszym przypadku filtr może działać na obszarze obrazu pomniejszonym o rozmiar filtru, jednak pozostaje pewna część obrazu leżąca na jego brzegu, która nie zostanie przetworzona, zatem jest to rozwiązanie w większości przypadków nieakceptowalne. Rozwiązaniem tego problemu jest rozszerzenie obrazu wejściowego do rozmiaru $I(M+S-1, N+T-1)$ i uzupełnienie powstałych brzegów w jeden z następujących sposobów:

1) stałą wartością dla każdego punktu leżącego poza brzegiem obrazu – Rysunek 5.3(a);
2) wartością punktu leżącego na granicy obrazu – Rysunek 5.3(b);
3) wartościami punktów z powielenia obrazu oryginalnego – Rysunek 5.3(c);
4) wartościami punktów z odbicia pionowego i poziomego obrazu – Rysunek 5.3(d).

Wartość stała na granicy jest najmniej wymagająca obliczeniowo ale wprowadza duże zakłamania na brzegach przefiltrowanego obrazu. Wypełnienie wartością brzegową lub powielenie obrazu na granicy wciąż może pozostawiać pewne przekłamania ale nie wprowadzają takiego błędu jak wartość stała. Są to stosunkowo najbardziej ekonomiczne sposoby rozszerza-



(a) (b)

(c) (d)

Rysunek 5.3. Uzupełnianie brzegów rozszerzonego obrazu o: (a) stałą wartość, (b) wartość brzegową, (c) powielenie obrazu na granicy, (d) odbicie obrazu na granicy.

nia granic obrazu. Najlepsze efekty ale najbardziej kosztowne daje odbicie poziome i pionowe obrazu na granicy.

Wszystkie powyższe rozważania dotyczyły obrazów w skali szarości, jednokanałowych. Dla obrazów wielokanałowych, kolorowych operacje filtracji najczęściej przeprowadza się dla każdego kanału osobno i niezależnie od pozostałych. Przy czym dla modeli separujących kanał jasności od chrominacji warto wprowadzić wagową filtrację z uwagi na dużo większe znaczenie kanału luminacji.

Właściwości danego filtru a co za tym idzie efekt filtracji zależy od definicji maski, czyli jej rozmiaru oraz wartości poszczególnych jej elementów składowych. O ile wielkość maski decyduje o sile danego filtru to wartości maski decydują o klasie filtru. Ogólnie można podzielić klasy filtru na rozmywające (dolnoprzepustowe) oraz wyostrzające (górnoprzepustowe).

5.1.1. Filtry rozmywające

Filtry rozmywające to filtry dolnoprzepustowe uśredniające wartości otoczenia znajdującego się w zasięgu maski filtru. Ideą takiego filtru jest zastąpienie wartości każdego punktu w obrazie przez średnią intensywność jego sąsiedztwa zdefiniowanego przez maskę filtru. Filtr taki tłumi w sygnale składowe widma o wysokiej częstotliwości a pozostawia bez zmian niskie częstotliwości. Najbardziej oczywistym zastosowaniem takiego filtru jest redukcja przypadkowego szumu i wygładzenie obrazu. Jednakże, krawędzie, które są prawie zawsze pożądaną informacją w obrazie również mają ostre przejścia jasności, zatem jako obszar o wysokiej częstotliwości ulegną tłumieniu przez filtr, który spowoduje ich rozmycie. Jest to niewątpliwie niepożądany efekt działania filtru i zarazem jego największa wada.

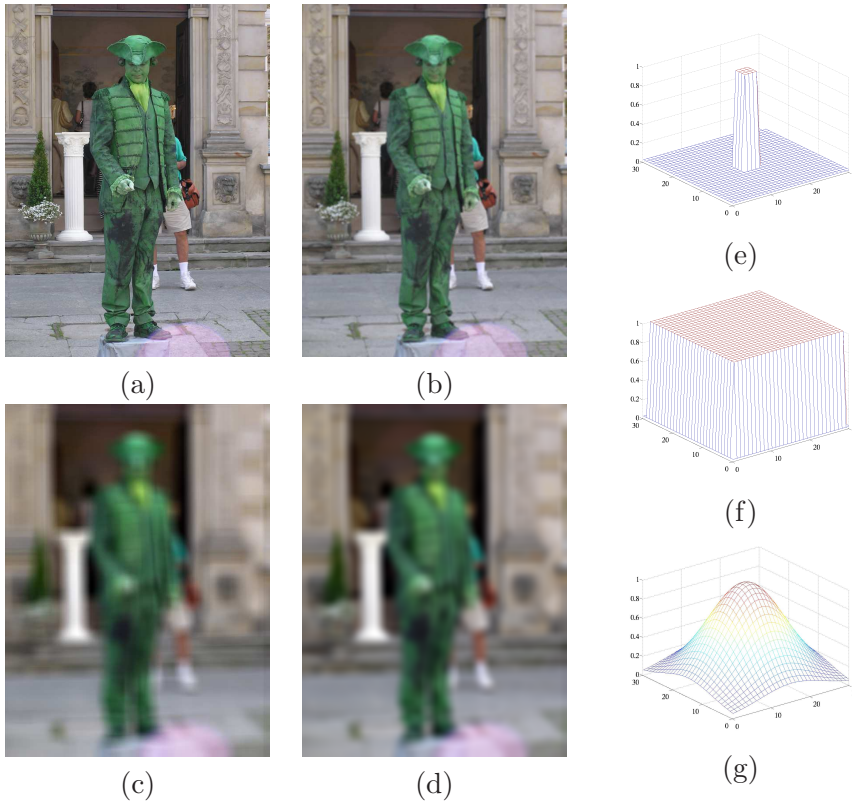
Najprostszy filtr rozmywający o rozmiarze 3×3 i współczynnikach równych 1 i wadze filtru równej 9 (przedstawiony na Rysunku 5.4 (e)) w wyniku działania zastępuje jasność punktu aktualnie przetwarzanego średnią arytmetyczną ze swojego 9-elementowego otoczenia. Tego typu filtr jedynkowy nazywa się filtrem uśredniającym lub pudełkowym (ang. *box filter*). Siłę takiego filtru można regulować w dwojaki sposób: pierwszy polega na zwiększeniu wielkości maski filtru (porównaj Rysunek 5.4 (b) i (c)), drugi na iteracyjnym, kilkukrotnym zastosowaniu danego filtru.

Kolejną wadą filtru uśredniającego jest jego "kwadratowość" objawiająca się pojawiającymi się na obrazie artefaktami (zobacz Rysunek 5.4 (c)). Zwiększenie wartości środkowego elementu maski pozwala zmniejszyć nieco negatywne skutki działania filtru dolnoprzepustowego zmniejszając równocześnie siłę filtru. Jeszcze lepszy efekt można osiągnąć aproksymując wartości maski filtru funkcją rozkładu Gaussa w dwuwymiarowej formie:

$$h(x, y) = e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{5.10}$$

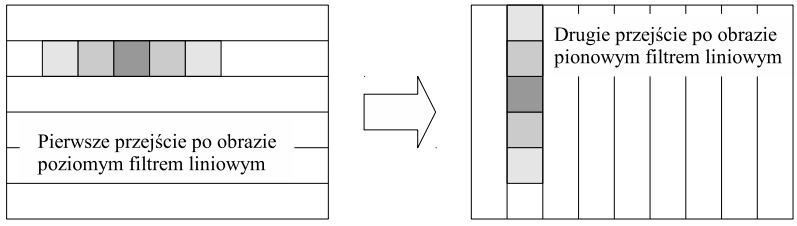
gdzie σ jest standardowym odchyleniem, x i y są wartościami całkowitymi, a współrzędna $(0, 0)$ jest w środku filtru. Przefiltrowany obraz oraz maska filtru przedstawione są na Rysunkach 5.4 odpowiednio (d) i (g).

Zaletami filtru uśredniającego i filtru Gaussa jest ich symetryczność pionowa i pozioma. Dzięki temu można mocno zredukować złożoność obliczeniową takiego filtru z kwadratowej do liniowej. W przypadku filtru pudełkowego dla pierwszego filtrowanego punktu obrazu wyznaczana jest średnia arytmetyczna wszystkich elementów z rozważanego otoczenia, ale przechodząc do następnego punktu obrazu wystarczy odjąć wartości punków z $x-2$ kolumny i dodać wartości punktów z $x+1$ kolumny do średniej. Jeszcze większe przyspieszenie można uzyskać wykorzystując fakt, że filtr jest jednakowy w kierunku x jak i w kierunku y, zatem można dany filtr (a raczej jego 1-wymiarowy odpowiednik) zastosować najpierw do wierszy, a następnie do



Rysunek 5.4. (a) Przykładowy obraz. (b) Obraz przefiltrowany prostym filtrem uśredniającym 3×3 oraz (e) maska filtru; (c) obraz przefiltrowany filtrem uśredniającym o promieniu $r = 15$ oraz (f) maska filtru; (d) obraz przefiltrowany filtrem Gaussa o promieniu $r = 15$ i $\sigma = 10$ oraz (g) maska filtru.

kolumn (zobacz Rysunek 5.5). Uzyskany efekt będzie identyczny z filtrem splotowym dwuwymiarowym, a koszt obliczeń znacząco maleje.



Rysunek 5.5. Schemat separacji splotu dwuwymiarowego na dwa jednowymiarowe następujące po sobie.

5.1.2. Filtry wyostrzające

Głównym celem filtrów wyostrzających jest podkreślenie przejść tonalnych intensywności punktów. Ten rodzaj filtracji używa filtrów górnoprzepustowych do wzmacniania wysokich częstotliwości widma sygnału i tłumienia niskich. O ile filtr rozmywający uśrednia wartości, co można porównać do całkowania, to filtr wyostrzający może być przyrównany do przestrzennego różniczkowania. Siła odpowiedzi na operator różniczkowania jest proporcjonalna do nieciągłości jasności w obrazie w tym punkcie, na którym działa operator. Stąd, różniczkowanie obrazu wzmacnia krawędzie i inne nieciągłości (również szum) a redukowana jest jasność w obszarach z wolno zmieniającymi się intensywnościami punktów.

W pierwszym przypadku rozpatrzmy pierwszą pochodną zdefiniowaną w kategoriach różnicowych:

$$\frac{\partial f}{\partial x} = f(x+1) - f(x) \tag{5.11}$$

W przypadku dwuwymiarowym wyrażenie to jest zastępowane gradientem funkcji $f(x, y)$:

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} f(x+1,y) - f(x,y) \\ f(x,y+1) - f(x,y) \end{bmatrix} \tag{5.12}$$

Aproksymując wartości pochodnych można stworzyć maskę filtru górnoprzepustowego. Najprostszym operatorem gradientowym jest operator Robertsa [38] uzyskany bezpośrednio ze wzoru 5.11 korzystający w zasadzie z maski o rozmiarze 2×2 o składowych (1) oraz (-1). Filtr ten wykrywa skok intensywności punktów w jednym z ośmiu możliwych kierunków. Główną jego wadą jest duża wrażliwość na szum oraz niska reakcja na właściwą krawędź. Dwa przykładowe warianty maski filtru Robertsa przedstawia Rysunek 5.6 (a).

Nieco lepsze efekty daje stosowania aproksymacji operatora gradientu na maskę o rozmiarze 3×3. Przykładem takiego operatora jest filtr Prewitta [36] pokazany w dwóch wariantach kierunkowych na Rysunku 5.6(b). Najczęściej jednak przy detekcji krawędzi stosuje się gradient Sobela [45] zdefiniowany następująco:

$$\begin{aligned} S_x = {} & [I(x+1,y+1) + 2I(x+1,y) + I(x+1,y-1)] - \\ & [I(x-1,y+1) + 2I(x-1,y) + I(x-1,y-1)] \\ S_y = {} & [I(x-1,y+1) + 2I(x,y+1) + I(x+1,y+1)] - \\ & [I(x-1,y-1) + 2I(x,y-1) + I(x+1,y-1)] \end{aligned}$$

0	0	0
0	1	-1
0	0	0

1	1	1
0	0	0
-1	-1	-1

1	2	1
0	0	0
-1	-2	-1

0	0	0
0	1	0
0	-1	0

(a)

1	0	-1
1	0	-1
1	0	-1

(b)

1	0	-1
2	0	-2
1	0	-1

(c)

Rysunek 5.6. Przykładowe maski 3×3 dla filtrów górnoprzepustowych. (a) operator Robertsa, (b) operator Prewitta, (c) operator Sobela. Górny wiersz pokazuje operatory dla gradientu pionowego, dolny poziomego.

dając maskę przedstawioną na Rysunku 5.6(c). Celem użycia wagi równej 2 w centralnych elementach maski jest uzyskanie pewnego rozmycia nadającego większe znaczenie centralnemu punktowi. Przykładowe obrazy przefiltrowane operatorami gradientowymi przedstawione są na Rysunku 5.7.

Niewątpliwą wadą filtrów gradientowych jest ich kierunkowość. Próbując zniwelować ten problem można posłużyć się drugą pochodną, tzw. laplasjanem:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{5.13}$$

zdefiniowanym w kategoriach różnicowych jako:

$$\begin{aligned} \frac{\partial^2 f}{\partial x^2} &= f(x+1,y) + f(x-1,y) - 2f(x,y) \\ \frac{\partial^2 f}{\partial y^2} &= f(x,y+1) + f(x,y-1) - 2f(x,y) \end{aligned} \tag{5.14}$$

Na podstawie (5.14) można stworzyć bezpośrednio symetryczną maskę filtru Laplace'a, np:

0	-1	0
-1	4	-1
0	-1	0

lub

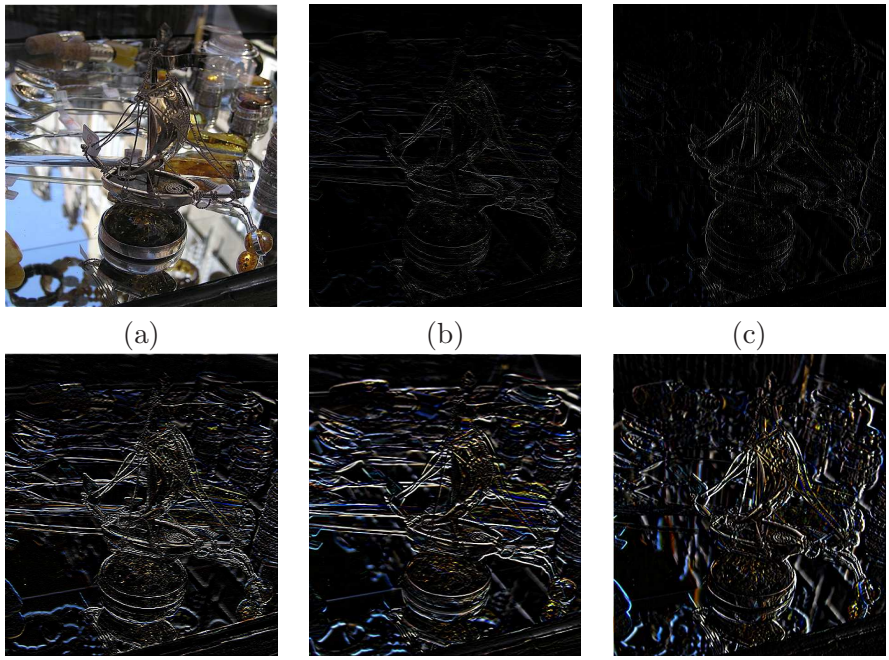
-1	-1	-1
-1	8	-1
-1	-1	-1

lub

-2	1	-2
1	4	1
-2	1	-2

Izotropowość (niezmienniczość względem rotacji) tego filtru zapewnia jednakową detekcję krawędzi bez względu na kierunek na jakim występują. Przykładowy obraz przefiltrowany filtrem Laplace'a jest pokazany na Rysunku 5.8 (d), (e), (f).

W obu przypadkach, pierwszej pochodnej i drugiej pochodnej, obrazy przefiltrowane mogą posiadać wartości ujemne z powodu ujemnych wartości



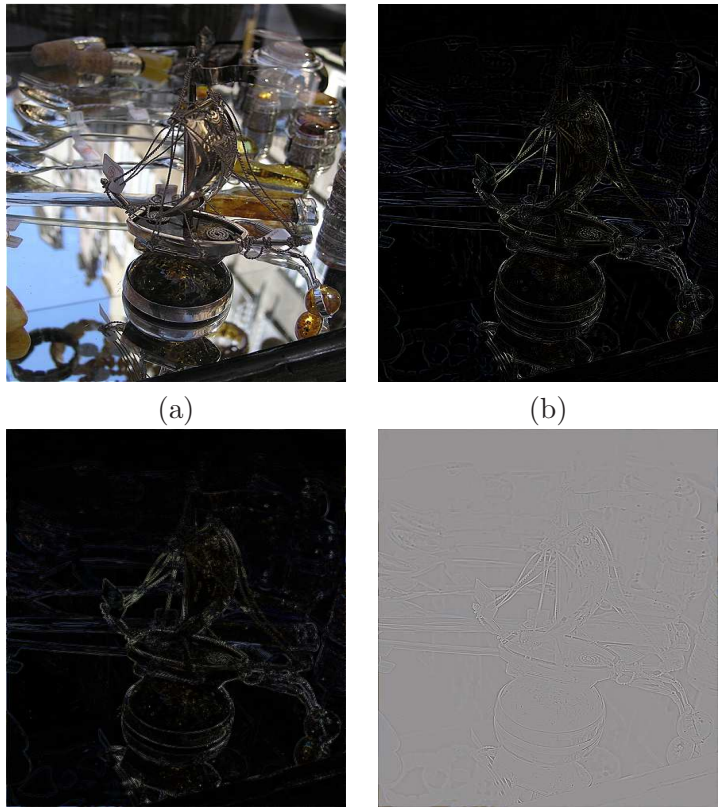
(a) (b) (c)

(d) (e) (f)

Rysunek 5.7. Przykłady filtrów gradientowych: (a) obraz oryginalny; (b) obraz przefiltrowany filtrem Robertsa pionowym; (b) obraz przefiltrowany filtrem Robertsa poziomym; (d) obraz przefiltrowany filtrem Prewitta pionowym; (e) obraz przefiltrowany filtrem Sobela pionowym; (f) obraz przefiltrowany filtrem Sobela poziomym;

w masce filtru. Jednakże, w przypadku obrazu cyfrowego wartości intensywności punktów muszą zawierać się w pewnym dodatnim przedziale. Zwykle obszary o wolno zmieniających się intensywnościach otrzymują wartości w okolicy zera, natomiast krawędzie i inne nieciągłości, w zależności od ich kierunku wartości dodatnie lub ujemne. Te wartości w najprostszym przypadku można przyciąć do wartości granicznych narzuconych przez definicję obrazu cyfrowego (zobacz Rysunek 5.8 (d)). Niestety tracimy wtedy sporo informacji o wielkości i kierunku skoku intensywności na krawędziach. Nieco lepszym rozwiązaniem będzie obliczenie wartości bezwzględnej intensywności punktu, dzięki czemu zachowamy przynajmniej informację o wielkości skoku (Rysunek 5.8 (e)). Trzecią możliwością jest przeskalowanie wartości obrazu przefiltrowanego do wartości z dziedziny obrazu (Rysunek 5.8 (f)).

Informację o krawędziach można wykorzystać do wyostrzania obrazu przez proste zsumowanie obrazu krawędzi z oryginalnym obrazem. Posłu-



(a) (b)

(c) (d)

Rysunek 5.8. (a) Przykładowy obraz oraz obraz przefiltrowany: (b) filtrem Laplace'a z przycięciem wartości mniejszych od zera i większych od maksymalnej wartości dopuszczalnej dla obrazu, (c) filtrem Laplace'a z wartością bezwzględną, (d) filtrem Laplace'a z pełnym skalowaniem wartości.

gując się filtrem Laplace'a wyostrzanie można zapisać jako:

$$g(x,y) = f(x,y) + \nabla^2 f(x,y) \tag{5.15}$$

Przykładowa maska filtru będzie zatem wyglądać następująco:

$$\begin{array}{|c|c|c|} \hline 0 & -1 & 0 \\ \hline -1 & 4 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} + \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 0 & -1 & 0 \\ \hline -1 & 5 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} \tag{5.16}$$

a efekt działania takiego filtru jest widoczny na Rysunku 5.9.

Suma dowolnego filtru górnoprzepustowego z obrazem oryginalnym da efekt "pozornego" wyostrzania obrazu. Dlaczego "pozornego" wyjaśni technika maski wyostrzającej.

(a) (b)

Rysunek 5.9. Przykład wyostrzania z zastosowaniem filtru Laplace'a. (a) Obraz oryginalny; (b) obraz przefiltrowany maską zdefiniowaną w 5.16.

Najpopularniejsza istniejąca technika wyostrzania wywodzi się jeszcze z poligrafii analogowej. W klasycznej ciemni fotograficznej aby uzyskać wyostrzoną odbitkę najpierw tworzyło się pomocniczy, celowo rozmyty negatyw. Z takiego negatywu robiono pozytyw i następnie naświetlano razem z oryginalnym negatywem. Tak powstały negatyw nazywany był maską wyostrzającą. Ostatnim krokiem było wspólne naświetlanie oryginalnego negatywu i maski wyostrzającej. Cała technika wyostrzania była nazywana maskowaniem wyostrzającym (ang. *unsharp masking*). Metodę tę stosuje się z powodzeniem w cyfrowej wersji przetwarzania obrazów. W najprostszej wersji składa się ona z dwóch głównych kroków:

1. Uzyskanie maski poprzez odjęcie od obrazu oryginalnego obrazu rozmytego filtrem dolnoprzepustowym Gaussa:

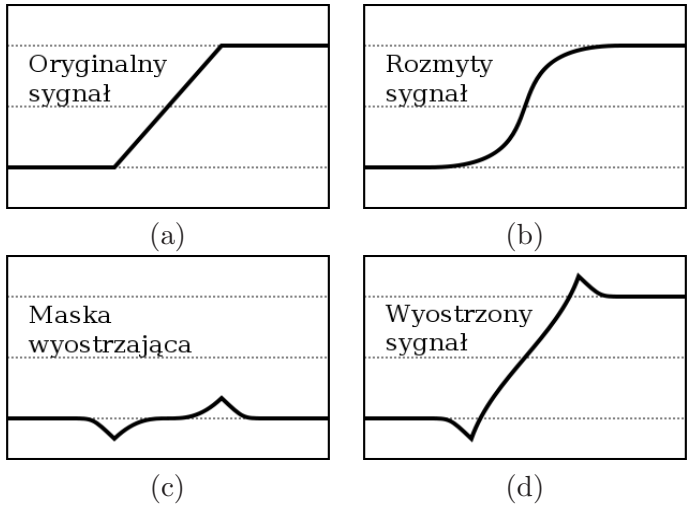
$$g_{mask}(x,y) = f(x,y) - G[f(x,y)] \tag{5.17}$$

2. Dodanie maski wyostrzającej z pewną wagą do obrazu oryginalnego:

$$g(x,y) = f(x,y) + \alpha * g_{mask}(x,y) \tag{5.18}$$

gdzie współczynnik $\alpha \geqslant 0$ nazywany wzmocnieniem decyduje o stopniu wyostrzenia obrazu.

Pierwszy punkt, czyli tworzenie maski można zastąpić dowolnym filtrem górnoprzepustowym jednak różnica filtru Gaussa i obrazu oryginalnego daje dużą swobodę regulacji parametrów filtru. Watro zaznaczyć, że nie jest



Rysunek 5.10. Ilustracja techniki maski wyostrzającej: (a) Oryginalny sygnał; (b) sygnał oryginalny po rozmyciu filtrem dolnoprzepustowym; (c) różnica sygnału oryginalnego i rozmytego, czyli maska wyostrzająca; (c) wyostrzony sygnał za pomocą sumy sygnału oryginalnego i maski wyostrzającej.

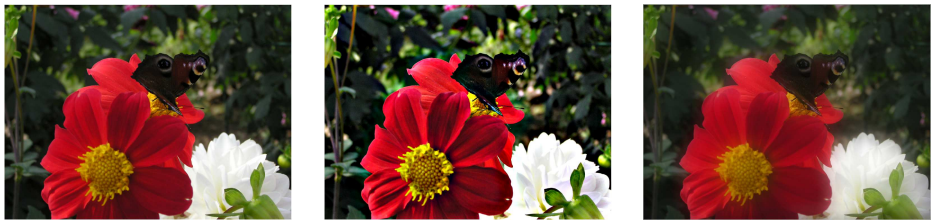
to rzeczywiste wyostrzanie obrazu a delikatne "oszustwo" polegające na lokalnym podbiciu kontrastu na krawędziach, tzn. punkty leżące po stronie ciemniejszej krawędzi staną się jeszcze ciemniejsze a punkty leżące po jaśniejszej stronie krawędzi staną się jaśniejsze. Przy niewielkim współczynniku wzmocnienia tego typu zmianę obrazu mózg ludzki interpretuje jako wyostrzenie. Niestety duże jego wartości wprowadzają już na tyle mocne zniekształcenie, że obraz jest wizualnie nieakceptowalny. Rysunek 5.10 obrazowo wyjaśnia sposób działania maski wyostrzającej a przykład obrazu uzyskanego przy użyciu tego filtru jest widoczny na Rysunku 5.11. Jeszcze lepsze efekty zwiększania ostrości obrazu można osiągnąć rezygnując z modelu RGB na rzecz modelu separującego luminację od kanałów chrominacji. Naturalnym wyborem będzie tutaj model CIE L*a*b* omawiany w rozdziale 4. Samemu wyostrzaniu będzie podlegał wtedy jedynie kanał luminacji, kanały chrominacji pozostają bez zmian. Możliwe jest wtedy stosowanie wyższych wartości współczynnika wzmocnienia α przy znacznie mniejszym poziome wprowadzanych szumów barwnych. Porównaj Rysunki 5.11 (c) i (e) oraz 5.11 (d) i (f).

Maskę wyostrzającą można wykorzystać również do wzmacniania lub osłabiania kontrastu bardziej globalnie. Daje to niewątpliwie daleko lepsze rezultaty niż liniowa regulacja kontrastu i jest z powodzeniem wykorzysty-

Rysunek 5.11. (a) Przykładowy obraz, (b) maska wyostrzająca dla parametrów filtru Gaussa: $r = 7, \sigma = 4.0$, (c) obraz wyostrzony ze współczynnikiem wzmocnienia $\alpha = 1$, (d) obraz wyostrzony ze współczynnikiem wzmocnienia $\alpha = 7$ - wyraźnie widoczny efekt podbicia lokalnego kontrastu na krawędziach i silne wzmocnienie szumu, (e) obraz wyostrzony z identycznymi parametrami jak w (c) ale wyostrzaniu podlegał jedynie kanał luminacji modelu CIE L*a*b*, (f) obraz wyostrzony z identycznymi parametrami jak w (d) ale wyostrzaniu podlegał jedynie kanał luminacji modelu CIE L*a*b*.

wana w programach do obróbki obrazu. Aby uzyskać taki efekt podbicia

kontrastu parametry maski wyostrzającej są ustawiane niejako na odwrót w stosunku do techniki wyostrzania, tj. promień filtru Gaussa powinien być stosunkowo duży (nawet do 10% wielkości obrazu), natomiast współczynnik wzmocnienia α mały. Przykładowe wartości parametrów takiej maski wyostrzającej oraz wzmocnione obrazy są przedstawione na Rysunku 5.12.



(a) (b) (c)

Rysunek 5.12. Przykład wykorzystania maski wyostrzającej do zmiany globalnego kontrastu obrazu: (a) oryginalny obraz, (b) obraz o zwiększonym kontraście, parametry filtru Gaussa to $r = 50, \sigma = 40$, (c) obraz o obniżonym kontraście z filtrem o tych samych parametrach co w (b) ale współczynnik α został przemnożony przez (-1).

5.2. Filtracja nieliniowa

5.2.1. Filtry statystyczne

Filtry statystyczne to nieliniowe przestrzenne filtry, których działanie jest oparte na kolejności wartości punktów zawartych w obrębie maski filtru. W wyniku filtracji aktualnie przetwarzany punkt obrazu zastępowany jest wartością wynikającą z miejsca w szeregu wartości. Sąsiedztwo danego punktu jest definiowane, analogicznie jak w przypadku filtrów splotowych. poprzez prostokątną maskę. Jednakże dopuszczalne wartości maski to jedynie wartości logiczne 0 i 1. Podczas filtracji punkt, który w odpowiadającej mu pozycji na masce ma wartość 0 nie jest brany pod uwagę przy obliczaniu wartości szukanego punktu. Zatem, pomimo prostokątnego kształtu maski, dzięki logicznym wartościom można definiować dowolny kształt takiego filtru. Najpopularniejszymi filtrami tej grupy są filtry minimum, maksimum oraz filtr medianowy.

5.2.1.1. Filtr minimum i maksimum

W obu przypadkach filtru minimum i maksimum wartość poszukiwana punktu wymaga ustalenia porządku wartości punktów z otoczenia zdefinio-

wanego przez maskę filtru, czyli posortowania ich względem wartości. Dla filtru minimum wartością szukaną jest najmniejsza wartość z posortowanego ciągu:

$$g(x,y) = \min_{(s,t)\in M} \{f(x,y)\} \tag{5.19}$$

gdzie M oznacza rozmiar maski filtru. Analogicznie dla filtru maksymalnego nową wartością jest największa wartość po uszeregowaniu otoczenia:

$$g(x,y) = \max_{(s,t)\in M} \{f(x,y)\} \tag{5.20}$$

Filtr minimalny jest często nazywany filtrem kompresującym albo erozyjnym ponieważ efektem jego działania jest zmniejszenie globalnej jasności obrazu, jasne obiekty ulegną zmniejszeniu a powiększą się obiekty ciemne. Filtr maksymalny bywa nazywany filtrem dekompresującym albo ekspansywnym gdyż w wyniku filtracji zwiększa globalnie jasność obrazu i analogicznie do filtru minimalnego tu powiększone zostaną obiekty jasne a zmniejszone ciemne.

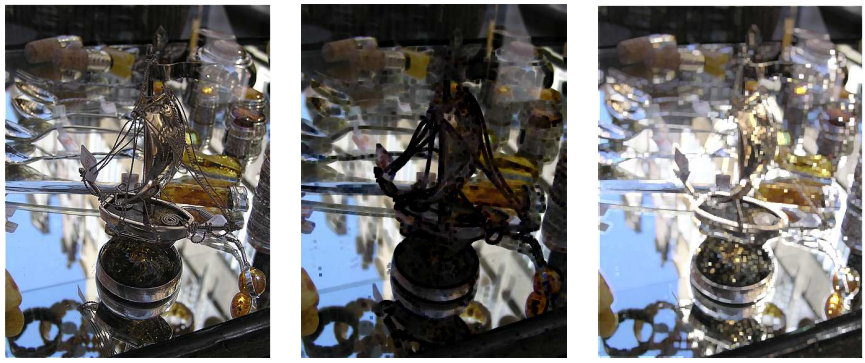
Filtry te bardzo często stosuje się łącznie. Na przykład, filtr minimum stosunkowo nieźle usuwa losowy szum ale za cenę zmniejszenia jasności obrazu. Zatem, żeby złagodzić niepożądany efekt na wynik filtracji stosuje się filtr maksymalny. Taka operacja jest często określana mianem zamknięcia. Analogicznie zastosowanie filtru maksymalnego i następnie filtru minimalnego spowoduje wzrost szczegółowości jasnych obiektów. Ta operacja nosi nazwę otwarcia. Zarówno filtr minimalny i maksymalny jak i operacja otwarcia i zamknięcia są rozszerzeniem na obrazy skali szarości morfologicznej operacji erozji i dylatacji dyskutowanej w Rozdziale 6. Rysunek 5.13 przedstawia przykłady filtracji minimalnej i maksymalnej.

5.2.1.2. Filtr medianowy

Filtr medianowy, bez wątpienia, obok filtru Gaussa, jest najważniejszym filtrem w przetwarzaniu obrazów cyfrowych. Zgodnie z nazwą po uszeregowaniu wartości punktów z otoczenia szukaną wartością jest mediana czyli wartość środkowa.

$$g(x,y) = \operatorname*{median}_{(s,t)\in M} \{f(x,y)\} \tag{5.21}$$

Filtr ten doskonale usuwa pewne typy szumów jednocześnie nie rozmywając obrazu w porównaniu do podobnej wielkości filtru rozmywającego. Inaczej mówiąc, filtr ten eliminuje z obrazu te punkty, których wartość znacznie odbiega od wartości otoczenia. Nie wprowadza również do obrazu żadnych nowych wartości. Co więcej jest stosunkowo bezpieczny dla krawędzi nie powodując ich degradacji. Filtr medianowy o dużym promieniu często jest



(a) (b) (c)

(d) (e)

Rysunek 5.13. Przykład filtracji minimalnej i maksymalnej: (a) oryginalny obraz, (b) obraz z zastosowaniem filtru minimum o promieniu $r = 2$, (c) obraz z zastosowaniem filtru maksimum o promieniu $r = 2$, (d) filtr zamknięcia o promieniu $r = 1$, (e) filtr otwarcia o promieniu $r = 1$.

używany do wygładzania obrazu a właśnie dzięki zachowaniu kontrastu na krawędziach jest znacznie bardziej użytecznym filtrem w porównaniu z filtrami dolnoprzepustowymi.

Naiwna implementacja filtru medianowego wymagałaby użycia algorytmu sortującego do znalezienia elementu środkowego. Najlepsze algorytmy sortujące ogólnego przeznaczenia, jak chociażby "Quick Sort" mają złożoność obliczeniową rzędu $O(n \cdot \log n)$ pomnożone przez ilość punktów w obrazie sprawia, że algorytmy tego typu nie specjalnie nadają się do wyznaczania mediany. Można nieco zmodyfikować algorytm szybkiego sortowania sprawiając, że algorytm kończy swoje działanie nie w chwili gdy wszystkie elementy są uszeregowane ale sporo wcześniej. Do wyboru mediany wystarczy żeby algorytm sortujący przetasował elementy zbioru tak aby na lewo od elementu środkowego znalazły się wartości mniejsze od środkowego a na

prawo elementy większe. Taki algorytm nosi nazwę Szybkiego Wyszukiwania (ang. *Quick Select*) i pomimo identycznej złożoności obliczeniowej jak Quick Sort jego wydajność praktyczne będzie większa.

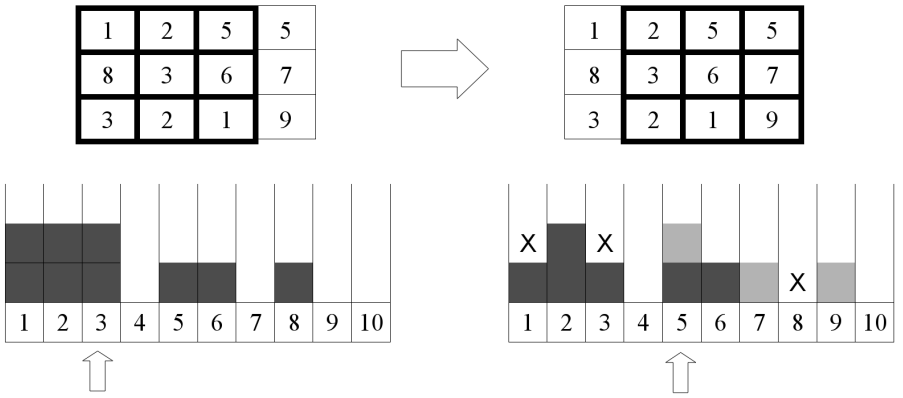
Obliczanie wartości mediany można jeszcze uprościć w przypadku małych promieni maski filtru. I tak, dla kwadratowego filtru o promieniu $r = 1$, tzn. wielkości 3×3 wartość mediany będzie równa środkowemu z 9 elementów i może być wyznaczona w następujący sposób:

$$\begin{array}{llll} sort(v_1, v_2); & sort(v_4, v_5); & sort(v_7, v_8); & sort(v_0, v_1); \\ sort(v_3, v_4); & sort(v_6, v_7); & sort(v_1, v_2); & sort(v_4, v_5); \\ sort(v_7, v_8); & sort(v_0, v_3); & sort(v_5, v_8); & sort(v_4, v_7); \\ sort(v_3, v_6); & sort(v_1, v_4); & sort(v_2, v_5); & sort(v_4, v_7); \\ sort(v_4, v_2); & sort(v_6, v_4); & sort(v_4, v_2); & \end{array}$$

gdzie zbiór $v_0, v_1, ..., v_8$ to zbiór wartości, dla którego należy znaleźć medianę, a funkcja $sort(a, b)$ zamienia miejscami wartości a z b jeżeli zachodzi $a > b$. Po wykonaniu takiej procedury wartość mediany będzie równa elementowi v_4. Filtr o wielkości $r = 2$ (rozmiar 5×5) wymagałby już 96 tego typu prostych sortowań.

Jeżeli założymy, że punkt obrazu może mieć wartość całkowitą z przedziału [0..255] wtedy jednym z najwydajniejszych sposobów wyznaczenia filtru medianowego jest użycie sortowania przez wstawianie. Złożoność obliczeniowa tego algorytmu sortującego $[O(n^2)]$ jest co prawda wyższa niż Quick Sortu ale możliwe jest wykorzystanie elementów przesortowanych dla danego punktu do znalezienia mediany w następnym punkcie obrazu. Dla przykładu, rozważmy obliczenie mediany dla maski 3×3 dla dwóch kolejnych punktów (Rysunek 5.14). Dla uproszczenia zakres wartości został ograniczony do [1..10].

Dla pierwszego punktu tworzony jest 10-elementowy koszyk, do którego będą wstawiane wartości obrazu objęte maską. W tym przypadku do pierwszej komórki koszyka wstawione zostały dwa elementy, ponieważ w sąsiedztwie centralnego punktu znajdują się dwie wartości równe 1. Dalej mamy 2 dwójki, 2 trójki i po jednym elemencie w piątej, szóstej i ósmej komórce. Ciemno szare pola wskazują ilość elementów o danej wartości. Wszystkich elementów jest 9, zatem środkowy element będzie piątym z kolei. Ten element jest w trzecim koszyku i taką też wartość ma mediana. W następnym kroku, po przejściu do kolejnego punktu obrazu z koszyka sortowania usuwane są elementy z kolumny, która po przesunięci maski znalazła się poza sąsiedztwem (zaznaczone znakiem X na rysunku), czyli wartości 1, 8 i 3 a dodane są nowe wartości z prawej kolumny (pola jasnoszare), czyli wartości 5, 7 i 9. Tym razem piąty element znajduje się w piątym koszyku. W ten sposób przesuwając się do kolejnych punktów obrazu w danym wierszu z koszyka sortowania usuwane są elementy jednej kolumny z lewej



Rysunek 5.14. Ilustracja wyznaczania filtru medianowego przy użyciu sortowania przez wstawianie. U góry wartości fragmentu obrazu. Pogrubione obramowanie oznacza wartości pokrywające się z maską filtru. U dołu koszyk sortowania dla odpowiedniego przypadku. Dokładny opis w tekście.

i dodawane nowe elementy kolumny z prawej. To podejście można rozszerzyć w kierunku pionowym o odejmowanie i dodawanie kolejnych wierszy maski przy przechodzeniu do kolejnych wierszy obrazu, co upraszcza całą złożoność obliczeniową filtru medianowego do niemalże liniowej. Przykłady filtracji medianowej przedstawione są na Rysunku 5.15.

(a) (b) (c)

Rysunek 5.15. Filtracja medianowa: (a) obraz zakłócony dosyć mocnym szumem monochromatycznym, (b) obraz po filtracji filtrem medianowym o promieniu $r = 1$, (c) obraz po filtracji filtrem medianowym o promieniu $r = 8$,.

5.2.1.3. Filtr punktu średniego

Filtr punktu średniego oblicza wartość leżącą w połowie pomiędzy wartością najmniejszą i największą dla danego otoczenia:

$$g(x,y) = \frac{1}{2}\left[\max_{(s,t)\in M}\{f(x,y)\} + \min_{(s,t)\in M}\{f(x,y)\}\right] \tag{5.22}$$

Filtr ten łączy cechy filtru statystycznego i uśredniającego. Bywa używany do redukcji szumu losowego o normalnym rozkładzie.

5.2.1.4. Filtry statystyczne a obrazy kolorowe

Dotychczasowe rozważania nad filtrami statystycznymi dotyczyły obrazów jednokanałowych. W przypadku obrazów wielokanałowych dany filtr można zastosować do każdego kanału osobno. Jednak wprowadza to nowe kolory do obrazu z powodu mieszania składowych z różnych punktów otoczenia. W momencie, gdy istotne jest jak najwierniejsze zachowanie kolorystyki filtry statystyczne mogą podejmować decyzje o wartości nowego punktu sortując luminację punktów i na tej podstawie wybierać punkt docelowy. Wymaga to jednak przechowywania w dodatkowej strukturze wszystkich składowych punktów otoczenia razem z wskaźnikami na ich szeregowane luminacje.

5.2.2. Filtry adaptacyjne

Filtry adaptacyjne to grupa filtrów, która zmienia swoje zachowanie w zależności od charakterystyki obrazu wewnątrz regionu zdefiniowanego w masce filtru. Zazwyczaj filtry te mają swój początek w typowych filtrach splotowych lub statystycznych i są odpowiednio modyfikowane w celu wzmocnienia działania podstawowego filtru i/lub osłabienia jego wad. Bardzo często ceną za taką poprawę jest złożoność filtru i co za tym idzie koszt obliczeniowy. Przedyskutujmy dwa przykłady tego typu filtrów: (1) adaptacyjny filtr medianowy i (2) filtr bilateralny.

5.2.2.1. Adaptacyjny filtr medianowy

Podstawowym filtrem w tym przypadku jest filtr medianowy, który dosyć dobrze radzi sobie z niezbyt silnym impulsowym szumem w obrazie. Modyfikacja tego filtru ma wzmocnić jego właściwości odszumiające oraz zapewnić zachowywanie detali podczas wygładzania szumu nieimpulsowego. Tak jak klasyczna mediana filtr ten działa na pewnym otoczeniu, jednak w tym przypadku maska filtru może się dynamicznie zmieniać w zależności od panujących warunków.

Działanie tego filtru można przedstawić w dwóch krokach: A i następującym po nim kroku B:

Krok A: $A1 = z_{med} - z_{min}$
$A2 = z_{med} - z_{max}$
If $A1 > 0$ AND $A2 < 0$, idź do Kroku B
Else zwiększ rozmiar maski
If rozmiar maski $\leqslant S_{max}$ powtórz Krok A
Else $f_{xy} = z_{med}$

Krok B: $B1 = z_{xy} - z_{min}$
$B2 = z_{xy} - z_{max}$
If $B1 > 0$ AND $B2 < 0$, $f_{xy} = z_{xy}$
Else $f_{xy} = z_{med}$

gdzie:

z_{min} = najmniejsza wartość w otoczeniu S_{xy}
z_{max} = największa wartość w otoczeniu S_{xy}
z_{med} = wartość mediany w otoczeniu S_{xy}
z_{xy} = wartość punktu o współrzędnych (x, y)
S_{max} = największa dopuszczalne wielkość maski
f_{xy} = wartość punktu wynikowego we współrzędnych (x, y)

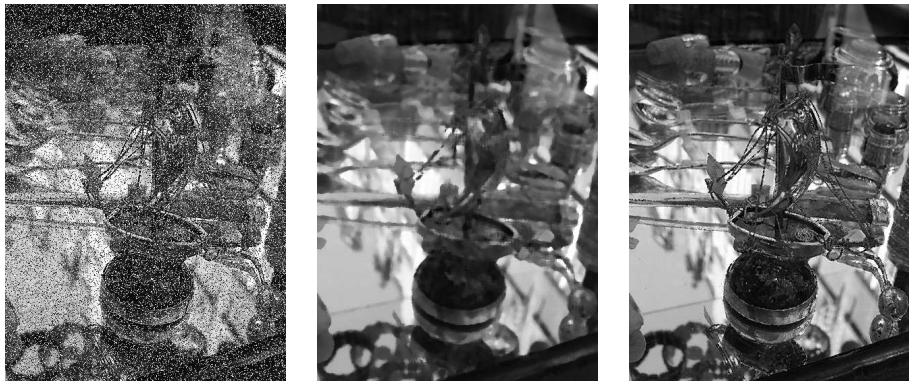
Wartości z_{min} oraz z_{max} są traktowane przez algorytm jako wartości szumu impulsowego nawet jeżeli nie są wartościami maksymalnymi lub minimalnymi w rozumieniu możliwych wartości punktu w obrazie. Celem kroku A jest określenie czy wartość wyjściowa mediany z_{med} jest impulsem czy nie. Jeżeli spełniony jest warunek $z_{min} < z_{med} < z_{max}$ wtedy z_{med} nie może być impulsem i algorytm przechodzi do kroku B. W kroku B sprawdzane jest czy wartość punktu z_{xy} jest impulsem w warunku $z_{min} < z_{xy} < z_{max}$. Jeżeli warunek jest prawdziwy (czyli z_{xy} nie jest impulsem) wtedy wartość wyjściowa tego punktu pozostaje niezmieniona i równa z_{xy}. Dzięki takiemu zachowaniu zniekształcenia typowe dla filtru medianowego zostają zredukowane. Jeżeli powyższy warunek jest fałszywy, tzn. $z_{xy} = z_{min}$ lub $z_{xy} = z_{max}$, oznacza to istnienie w tym punkcie silnego impulsu a wartość docelowa punktu zostaje zastąpiona przez wartość mediany otoczenia z_{med}, która, jak wiadomo z kroku A, sama nie jest szumem impulsowym. Ostatni etap jest klasyczną medianą, która, gdyby nie warunki testujące występowanie szumu, modyfikowałaby każdy punkt obrazu, powodując niepotrzebną utratę detali.

Przypuśćmy jednak, że w krok A znalazł wartość impulsową, czyli nie przeszedł do kroku B. Wtedy algorytm zwiększa rozmiar maski i ponawia wykonywanie kroku A. Pętla trwa dopóki algorytm albo przejdzie do kroku B albo rozmiar maski urośnie do narzuconej odgórnie maksymalnej wielkości maski. W tym przypadku wartość punktu docelowego zostaje zastąpiona

wartością mediany otoczenia z_{med}, przy czym nie ma gwarancji, że ta wartość nie jest impulsem.

Za każdym razem gdy algorytm znajdzie wartość docelową f_{xy}, maska filtru S_{xy} jest przenoszona do następnego punktu obrazu a algorytm jest stosowany do tego punktu z początkowymi wartościami.

Efektem stosowania adaptacyjnego filtru medianowego będzie usuwanie impulsowego szumu (typu 'sól i pieprz'), wygładzanie szumu innego pochodzenia niż impulsowy oraz jak największe zachowanie detali obrazu. Porównanie zwykłego filtru medianowego i jego adaptacyjnej wersji zostało przedstawione na Rysunku 5.16.



(a) (b) (c)

Rysunek 5.16. Porównanie działania zwykłego filtru medianowego i jego adaptacyjnej wersji na mocno zaszumionym obrazie (a); (b) obraz przefiltrowany zwykłym filtrem medianowym o rozmiarze 7×7; (c) obraz przefiltrowany filtrem adaptacyjnym medianowym o $S_{max} = 7$.

5.2.2.2. Filtr bilateralny

Filtr bilateralny jest techniką nieliniowego filtrowania [51]. Celem filtru jest przekształcenie obrazu identyczne z filtrem dolnoprzepustowym ale tylko w obszarach o niewielkiej wartości gradientu. Obszary o dużych skokach gradientu mają być przenoszone w oryginalnej, niezmienionej postaci. Taki filtr ma wygładzać obraz z równoczesnym zachowywaniem krawędzi. Zaproponowany filtr bilateralny jest rozwinięciem filtru dolnoprzepustowego, zatem i w tym przypadku filtr działa lokalnie na pewnym otoczeniu uśredniając je ale maska filtru jest liczona dynamicznie dla każdego punktu z uwzględnieniem dodatkowych założeń. Wyjściowe wartości maski określane klasycznie w funkcji odległości od środka filtru są dodatkowo przemnażane przez funkcję intensywności punktów z rozważanego otoczenia. Innymi słowy, dany punkt z sąsiedztwa wnosi tym większą wagę do maski im mniej-

sza jest jego odległość do centrum, zarówno w geometrycznym rozumieniu jak też jego podobieństwa w sensie intensywności. Punkty, których wartość mocno odbiega od wartości punktu środkowego filtru wnoszą mniejszą wagę, nawet jeżeli leżą w pobliżu punktu centralnego. Samo obliczanie współczynników filtru odbywa się to przez połączenie dwóch filtrów, jednego w dziedzinie przestrzennej i jednego w dziedzinie intensywności.

Prosty ale istotny przypadek filtru bilateralnego polega na użyciu funkcji rozkładu Gaussa dla obu przypadków: funkcji odległości i funkcji intensywności z euklidesową odległością pomiędzy jego argumentami. Funkcja odległości h_d będzie miała postać:

$$h_d(x_0 - x) = e^{-\frac{1}{2}\frac{d(x_0 - x)^2}{\sigma_d^2}} \tag{5.23}$$

gdzie x_0 określa położenie centralnego punktu maski a $d(x_0 - x)$ jest euklidesową odległością pomiędzy rozważanym punktem x a x_0. Funkcja intensywności h_I będzie miała postać:

$$h_I(x_0 - x) = e^{-\frac{1}{2}\frac{\delta(f(x_0) - f(x))^2}{\sigma_I^2}} \tag{5.24}$$

gdzie $\delta(f(x_0) - f(x))$ jest różnicą intensywności punktów x i x_0. Ostateczna maska filtru jest konstruowana przez pomnożenie wartości funkcji odległości i funkcji intensywności. Sama procedura filtracji będzie już zwykłym splotem funkcji obrazu f z funkcją maski $h_d \times h_I$:

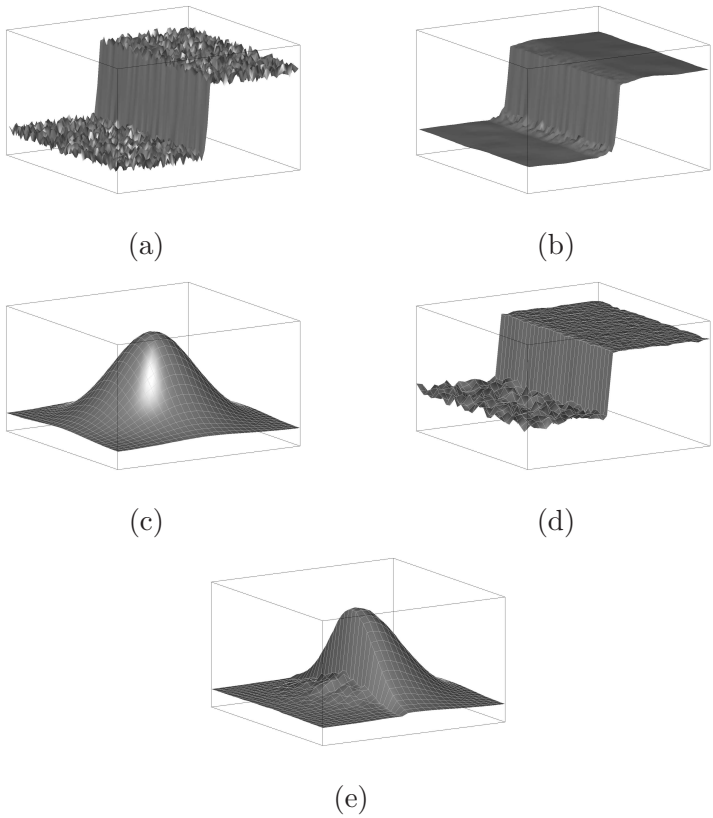
$$f(x_0) = \frac{1}{k}\sum_{i \in R} f(x_i) \times h_d(x_0 - x_i) \times h_I(x_0 - x_i) \tag{5.25}$$

gdzie R jest otoczeniem punktu x_0 branym pod uwagę przy filtracji, k jest współczynnikiem normalizacyjnym równym:

$$k = \sum_{i \in R} h_d(x_0 - x_i) \times h_I(x_0 - x_i) \tag{5.26}$$

Rysunek 5.17 przedstawia schematycznie sposób konstrukcji i działania filtru bilateralnego. Na Rysunku 5.17(a) widoczna jest krawędź i lekko zaszumione otoczenie. Sam filtr będzie wyśrodkowany na punkcie o dużej wartości (jasny) tuż przy krawędzi. Rysunek 5.17(c) przedstawia przestrzenną część maski z typowym rozkładem Gaussa w funkcji odległości od punktu centralnego. Rysunek 5.17(d) pokazuje część maski związaną z intensywnościami punktów. Wyraźnie widać, że punkty leżące po drugiej stronie krawędzi mają wartości znacznie mniejsze od wartości punktów leżących po tej samej stronie krawędzi. W rezultacie filtr zastępuje jasny punkt w centrum

przez uśrednione wartości punktów z jasnego otoczenia a niemalże ignoruje ciemne punkty. Na Rysunku 5.17(e) widoczna jest gotowa maska filtru dla tego konkretnego punktu. Przefiltrowane otoczenie rozważanej krawędzi jest zilustrowane na Rysunku 5.17(b).



Rysunek 5.17. Ilustracja działania filtra bilateralnego: (a) krawędź w obrazie z niewielkim losowym szumem, (b) krawędź po filtracji, szum został stłumiony przy zachowaniu skoku intensywności na krawędzi, (c) część przestrzenna maski filtru h_d, (d) część z dziedziny intensywności filtru h_I, (e) maska filtru dla środkowego punktu obrazu $h = h_d \times h_I$.

Rysunek 5.18 przedstawia obraz poddany filtracji dla różnych wartości parametrów funkcji odległości σ_d i funkcji intensywności σ_I. Na rysunku tym widać potencjał filtru zwłaszcza przy usuwaniu informacji o teksturze. Pewne "uproszczanie" obrazu widoczne na Rysunkach 5.18 (e) i (f) jest bardzo użyteczne przy redukcji danych w obrazie ale bez straty ogólnych kształtów i konturów. Dużo silniejszy efekt rozmycia (wygładzenia), przy

jednoczesnym skutecznym zachowaniu krawędzi daje iteracyjne użycie filtru z niewielkimi parametrami σ_d i σ_I (zobacz Rysunek 5.18 (g) i (h)).

Zastosowanie funkcji rozkładu Gaussa jako funkcji bazowych dla dziedziny przestrzennej i dziedziny intensywności nie jest jedynym możliwym wyborem, jednakże, z racji swojego niezmienniczego charakteru jest to dosyć naturalny wybór. W pozycji [30] czytelnik znajdzie szersze omówienie filtru bilateralnego w kontekście odszumiania obrazów cyfrowych razem z przykładami wydajnej implementacji.

(a)

(b)

(c)

(d)

(e)

(f)

Rysunek 5.18. Filtracja bilateralna: (a) obraz oryginalny oraz obrazy po filtracji ze współczynnikami: (b) $\sigma_d = 5$, $\sigma_I = 0.1$, (c) $\sigma_d = 15$, $\sigma_I = 0.1$, (d) $\sigma_d = 5$, $\sigma_I = 0.5$, (e) $\sigma_d = 5$, $\sigma_I = 1$, (f) $\sigma_d = 3$, $\sigma_I = 0.04$ po 10 iteracjach.

Rozdział 6

Przekształcenia morfologiczne obrazów

Morfologia, ogólnie to nauka o strukturze, kształcie i budowie. W przetwarzaniu obrazów stosujemy naukę zwaną morfologią matematyczną [1] jako narzędzie służące do wydobywania z obrazów elementów użytecznych w reprezentacji i opisie kształtów, takich jak kontur, szkielet czy otoczka. Metody morfologii stosuje się również w pre- i postprocesingu do filtracji, pogrubiania lub przycinania. Przykładami konkretnego zastosowania metod morfologicznych może być algorytm zliczający ilość wystąpień cząstek w polu widzenia, weryfikacja poprawności wykonania ścieżek na płytce obwodu drukowanego czy przedstawienie w formie szkieletowej danego obiektu na obrazie.

Idea morfologii matematycznej została opracowana i rozwinięta bazując na algebrze Minkowskiego [27] przez Matherona [26] i Serrę [41] w latach 60-tych 20 wieku. Oni też po raz pierwszy zastosowali tę naukę w przetwarzaniu obrazów cyfrowych. Steinberg w [47] po raz pierwszy zastosował metody morfologii do przetwarzania obrazów medycznych i przemysłowych.

Przekształcenie morfologiczne, podobnie jak filtry przestrzenne, uwzględniają otoczenie analizowanego punktu, ale w przeciwieństwie do filtrów operacje morfologiczne zachodzą wtedy gdy spełniony jest określony warunek logiczny. Odpowiednikiem maski z filtracji przestrzennej w morfologii matematycznej jest element strukturalny (ang. *structing element (SE)*) zwany czasami wzorcem. Jest to w ogólności mały obraz binarny stosowany do badania aktualnie przetwarzanego punktu. Wartości niezerowe elementu strukturalnego definiują jego kształt i jako jedyne biorą udział w obliczeniach morfologicznych. Dodatkowo element strukturalny musi posiadać punkt "centralny" (ang. *origin point*), który jest środkiem lokalnego układu współrzędnych danego wzorca. W przypadku symetrycznego elementu strukturalnego zakłada się, że punkt centralny znajduje się w centrum symetrii układu, w innym przypadku punkt centralny musi być wskazany osobno. Wybierając odpowiedni kształt elementu strukturalnego operacje morfologiczne stają się wrażliwe na konkretne kształty w obrazie.

W poniższych podrozdziałach najpierw zostaną opisane podstawy morfologi matematycznej dla obrazów binarnych. Następne podrozdziały zilustrują konkretne algorytmy przetwarzające obrazy przy użyciu morfologii matematycznej. Ostatni rozdział rozszerza stosowanie morfologii matematycznej na obrazy w skali szarości.

[1] Dział matematyki oparty na teorii zbiorów i topologii służący do analizy i przetwarzania struktur geometrycznych.

6.1. Podstawy morfologii

6.1.1. Erozja i Dylatacja

Erozja jest podstawową operacją morfologiczną. Formalnie, erozja zbioru A za pomocą wzorca B jest zdefiniowana:

$$A \ominus B = \{z|(B)_z \in A\} \tag{6.1}$$

To równanie oznacza, że erozja A za pomocą B jest zbiorem wszystkich punktów z takich, że B przesunięte o z jest zawarte w A. W kontekście obrazu binarnego, niech zbiór A będzie zbiorem wszystkich punktów o wartości 1 a zbiór B elementem strukturalnym. Powyższe równanie oznacza taki wynikowy obraz binarny, w którym dla każdego analizowanego punktu element strukturalny B musi w całości zawierać się w zbiorze A. Innymi słowy, element strukturalny B nie może mieć żadnych wspólnych elementów z tłem obrazu (punktami obrazu o wartości 0). Z tak sformułowanej definicji równanie 6.1 można zapisać równoważnie jako:

$$A \ominus B = \{z|(B)_z \cap A^C = \phi\} \tag{6.2}$$

gdzie A^C jest dopełnieniem A, ϕ jest zbiorem pustym. Efektem zastosowania erozji nad obrazem będzie usunięcie punktów na krawędziach obiektów lub całkowite usunięcie obiektów, które są mniejsze niż element strukturalny. Przykład erozji morfologicznej jest przedstawiony na Rysunku 6.1. Przyjęto konwencję, że punkt czarny jest binarną jedynką a punkt biały binarnym zerem.

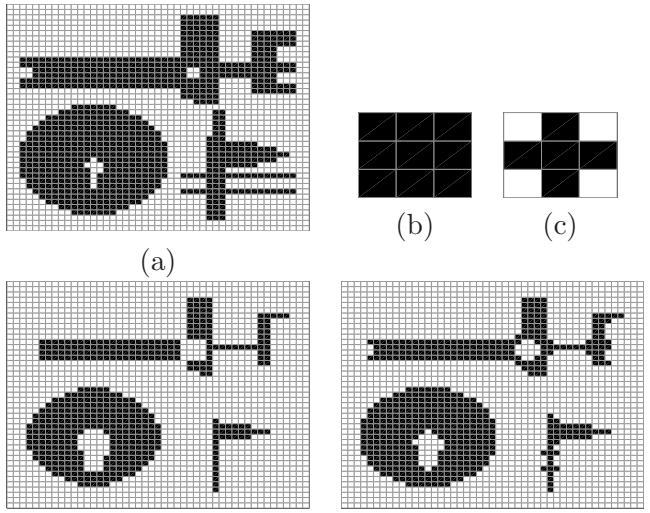
Dylatacja (czasami jest używana nazwa dylacja) A za pomocą elementu strukturalnego B może zostać zdefiniowana w następujący sposób:

$$A \oplus B = \{z|(\hat{B})_z \cap A \neq \phi\} \tag{6.3}$$

gdzie $\hat{B}$ jest odbiciem B: $\hat{B} = \{z|z = -b, \text{dla } b \in B\}$, tzn. $\hat{B}$ jest zbiorem punktów w B, których współrzędna (x) została zamieniona na przeciwną $(-x)$. Dylatacja A z elementem strukturalnym B jest zbiorem takich przemieszczeń z, że $\hat{B}$ i A mają co najmniej jeden punkt wspólny. W oparciu o tę interpretację można definicje dylatacji podać w następujący sposób:

$$A \oplus B = \{z|[(\hat{B})_z \cap A] \in A\} \tag{6.4}$$

W przeciwieństwie do erozji dylatacja powoduje rozrost (pogrubianie) obiektów w obrazie binarnym. Sposób w jaki będzie pogrubiony obiekt jest kontrolowany przez kształt elementu strukturalnego. Na Rysunku 6.2 pokazane zostały dwa elementy strukturalne i efekt zastosowania ich dla obrazu binarnego.



Rysunek 6.1. Przykład erozji morfologicznej. (a) binarny obraz, (b) i (c) element strukturalny, odpowiednio kwadratowy, wielkości 3×3 oraz krzyż 3×3, nie jest zachowana skala pomiędzy obrazem a obiektem strukturalnym, (d) obraz po erozji elementem strukturalnym (b), (e) obraz po erozji elementem strukturalnym (c).

W obu przypadkach $\hat{B} = B$ ponieważ element strukturalny jest symetryczny względem swojego centralnego punktu. Jednym z najprostszych zastosowań dylatacji jest tworzenie połączeń pomiędzy obiektami i wypełnienie dziur i zatok w obiektach.

Erozja i dylatacja są dualne względem siebie, to znaczy:

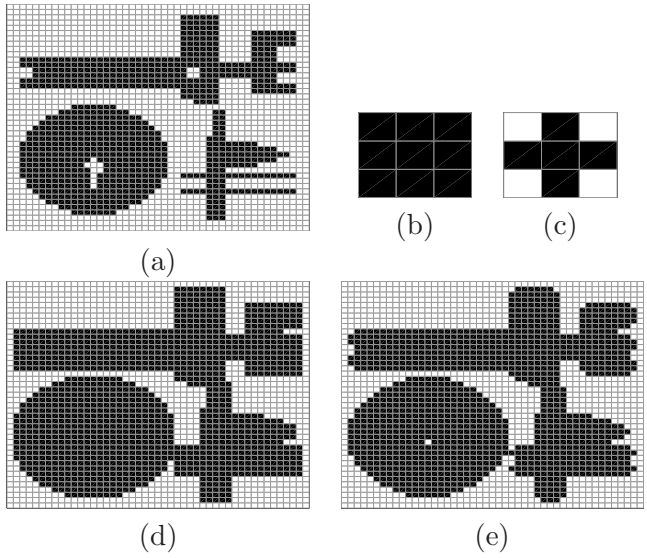
$$(A \ominus B)^C = A^C \oplus \hat{B} \tag{6.5}$$

oraz

$$(A \oplus B)^C = A^C \ominus \hat{B} \tag{6.6}$$

Ten dualny charakter jest szczególnie przydatny, gdy element strukturalny jest symetryczny względem swojego centrum. Wtedy erozja A elementem strukturalnym B jest równoważna dylatacji tła A (czyli dopełnienia A^C) tym samym elementem strukturalnym. Analogicznie znajduje to zastosowania w przypadku równania 6.6.

Zarówno erozję jak i dylatację można wykonywać iteracyjnie na wyniku danej operacji morfologicznej. I tak $(A \ominus kB) = (..((A \ominus B).. \ominus B)$ będzie wielokrotną operacją erozji prowadzącą do k krotnego zmniejszenia obiektu, a $(A \oplus kB) = (..((A \oplus B).. \oplus B)$ będzie wielokrotną operacją dylatacji, prowadzącą do k krotnego przyrostu konturu obiektu.



Rysunek 6.2. Przykład dylatacji morfologicznej. (a) binarny obraz, (b) i (c) element strukturalny, odpowiednio kwadratowy, wielkości 3×3 oraz krzyż 3×3, nie jest zachowana skala pomiędzy obrazem a obiektem strukturalnym, (d) obraz po dylatacji elementem strukturalnym (b), (e) obraz po dylatacji elementem strukturalnym (c).

Erozja i dylatacja są najbardziej podstawowymi operacjami morfologicznymi a zarazem najważniejszymi. W bardzo wielu przypadkach algorytmy morfologiczne omawiane w następnym podrozdziale są kombinacją właśnie erozji i dylatacji.

6.1.2. Otwarcie i zamknięcie

Jak wykazano w poprzednim podrozdziale erozja pomniejsza obiekt a dylatacja go powiększa. Operacje te można połączyć ze sobą wykonując jedną nad drugą. W zależności od kolejności operacji uzyskane transformacje nazywają się otwarciem i zamknięciem. Otwarcie zbioru A przez strukturalny element B jest zdefiniowane jako:

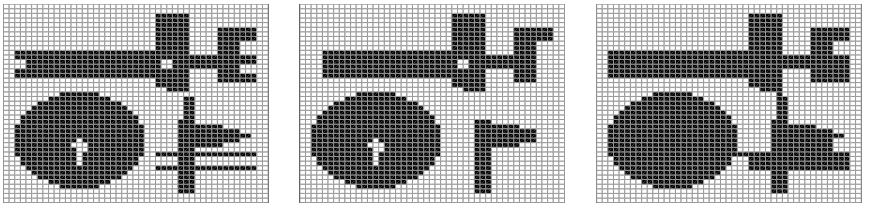
$$A \circ B = (A \ominus B) \oplus B \tag{6.7}$$

Z tego równania wynika, że otwarcie A przez B jest erozją A przez B i następnie dylatacją wyniku przez B. Podobnie zdefiniowane jest zamknięcie zbioru A przez element strukturalny B:

$$A \bullet B = (A \oplus B) \ominus B \tag{6.8}$$

Zatem zamknięcie A przez B jest dylatacją A przez B i następnie erozją wyniku przez B.

Operacja otwarcia usuwa z obiektu drobne szczegóły, może przerywać cienkie połączenia między obiektami, zewnętrzne ostre narożniki zostają wygładzone, podczas gdy wewnętrzne narożniki pozostają bez zmian. Analogicznie, w przypadku zamknięcia może powodować połączenie obiektów znajdujących się blisko siebie, niewielkie dziury są wypełniane, wewnętrzne narożniki zostają wygładzone, natomiast zewnętrzne pozostają bez zmian. Efekty otwarcia i zamknięcia zilustrowane są na Rysunku 6.3.



(a) (b) (c)

Rysunek 6.3. Morfologiczne otwarcie i zamknięcie: (a) binarny obraz, (b) obraz po otwarciu przez element strukturalny z Rysunku 6.2(b), (b) obraz po zamknięciu przez element strukturalny z Rysunku 6.2(b),

Tak jak w przypadku erozji i dylatacji, otwarcie i zamknięcie są dualne względem siebie, to znaczy:

$$(A \bullet B)^C = (A^C \circ \hat{B}) \tag{6.9}$$

oraz

$$(A \circ B)^C = (A^C \bullet \hat{B}) \tag{6.10}$$

Dodatkowo, operacja otwarcia ma następujące własności:
— wynik operacji otwarcia zawiera się w zbiorze A, tzn: $A \circ B \in A$;
— jeżeli C jest podzbiorem D, wtedy $C \circ B$ jest podzbiorem $D \circ B$;
— $(A \circ B) \circ B = A \circ B$, czyli wielokrotne otwarcie daje rezultat identyczny z pierwszym rezultatem (idempotentność).

Analogicznie, operacja zamknięcia ma następujące własności:
— A zawiera się wyniku operacji zamknięcia, tzn: $A \in A \bullet B$;
— jeżeli C jest podzbiorem D, wtedy $C \bullet B$ jest podzbiorem $D \bullet B$;
— $(A \bullet B) \bullet B = A \bullet B$, czyli wielokrotne zamknięcie daje rezultat identyczny z pierwszym rezultatem (idempotentność).

Ostatnia własność otwarcia i zamknięcia jasno mówi, że iteracyjne powtarzanie danej operacji nie zmienia wcześniejszego rezultatu. Ta własność odróżnia te operacje od erozji i dylatacji, dla których wynik był addytywny.

Morfologiczne operacje otwarcia i zamknięcia mogą być wykorzystywane do konstrukcji filtrów podobnych do filtrów przestrzennych dyskutowanych w rozdziale 5. Morfologiczny filtr składający się z otwarcia i następnie zamknięcia eliminuje przypadkowy szum nie zniekształcając zbyt mocno istotnych obiektów. Operacja otwarcia usuwa drobne elementy takie jak szum ale również może powodować przerywanie się istotnych obiektów. Następująca po niej operacja zamknięcia ma usunąć ten niepożądany efekt przez domknięcie powstałych dziur i połączenie w przewężeniach. Operacja zamknięcia a następnie otwarcia może zastępować filtr wykrywania obszarów o konkretnej fakturze. Poprzez odpowiedni dobór elementu strukturalnego operacja zamknięcia powoduje całkowite wypełnienie obszaru o szukanej fakturze. Następująca po niej operacja otwarcia pozostawi na obrazie tylko duże, "zalane" obszary, usuwając resztę. Jeżeli wynik tej operacji zostanie złożony z obrazem wejściowym to w rezultacie pozostaną tylko obszary o poszukiwanej fakturze.

6.1.3. Trafiony-chybiony (ang. *Hit-and-miss*)

Morfologiczna operacja Trafiony-chybiony jest podstawowym narzędziem w detekcji kształtów i pozwala dzięki odpowiednio zdefiniowanemu elementowi strukturalnemu wykryć szukany obiekt w obrazie. Co więcej stanowi ona pewnego rodzaju uogólnienie erozji i dylatacji, które można wyprowadzić jako szczególne przypadki transformacji hit-and-miss. Uogólniony jest również element strukturalny składający się z dwóch rozłącznych zbiorów: $B = (B_1, B_2)$ takich, że element strukturalny B_1 ma trafiać w szukany obiekt A, a element strukturalny B_2 ma trafiać w tło A^C, czyli chybiać obiekt A. Podsumowując, operację hit-and-miss można zdefiniować w następujący sposób:

$$A \circledast B = \{z | B_1 \in A \ \cap B_2 \in A^C\} \tag{6.11}$$

Korzystając z definicji operacji erozji transformację trafiony-chybiony można zdefiniować równoważnie jako:

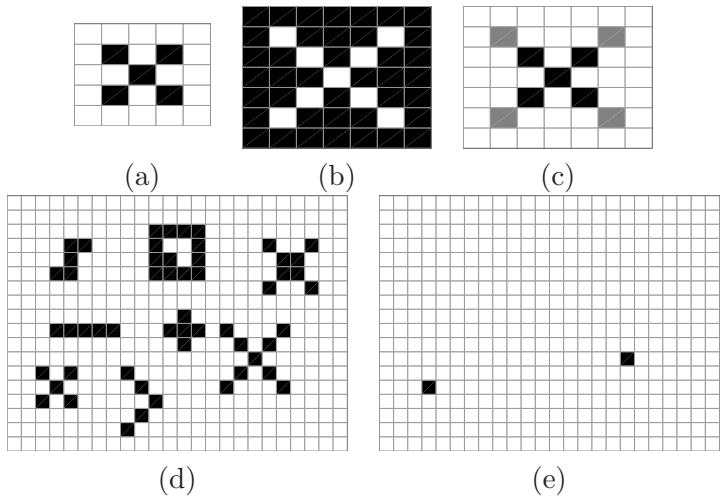
$$A \circledast B = (A \ominus B_1) \cap (A^C \ominus B_2) \tag{6.12}$$

lub, korzystając z dualnych relacji erozji i dylatacji (równania 6.5 i 6.6) za pomocą erozji i dylatacji:

$$A \circledast B = (A \ominus B) - (A \oplus \hat{B_2}) \tag{6.13}$$

Korzystając z równań 6.11 i 6.13 można z definicji operacji trafiony-chybiony wyprowadzić definicję erozji poprzez podstawienie pustego elementu strukturalnego $B_2 = \phi$. Dylatacja jest wtedy automatycznie definiowana przez własność dualności wyrażoną w równaniu 6.6.

Prześledźmy na przykładzie wykrywanie w obrazie binarnym znaków X o wielkości od 3×3 do 5×5 punktów. Elementem strukturalnym B_1, który ma zawsze trafić z szukany obiekt A będzie kształt pokazany na Rysunku 6.4(a). Jest to część wspólna znaków X o wielkości 3×3 i 5×5 punktów. Elementem strukturalnym B_2, który ma zawsze trafić w dopełnienie obiektu A^C, czyli tło będzie kształt z Rysunku 6.4(b). W ogólności element strukturalny $B = (B_1, B_2)$ można przedstawić jak na Rysunku 6.4(c), gdzie kolor czarny oznacza trafienie zawsze w szukany kształt, biały trafienie zawsze w tło a szary - element dowolny nie biorący udziału w operacji.



Rysunek 6.4. Ilustracja operacji trafiony-chybiony. (a) element strukturalny B_1 trafiający w szukany obiekt A, (b) element strukturalny B_2 trafiający w tło A^C, (c) wspólna notacja element strukturalnego $B = (B_1, B_2)$ – kolor czarny oznacza trafienie w szukany obiekt, biały trafienie w tło, szary jest nie brany pod uwagę przy obliczeniach, (d) obraz binarny początkowy, (e) obraz binarny z wykrytymi obiektami o szukanym kształcie.

6.2. Algorytmy morfologiczne

6.2.1. Ekstrakcja konturu

Wyznaczanie konturu (ang. *boundary extraction)* obiektu A oznaczanego przez $\kappa(A)$, w obrazie binarnym może zostać osiągnięte dzięki erozji zbioru A przez element strukturalny B i następnie odjęciu wyniku od oryginalnego zbioru, tzn:

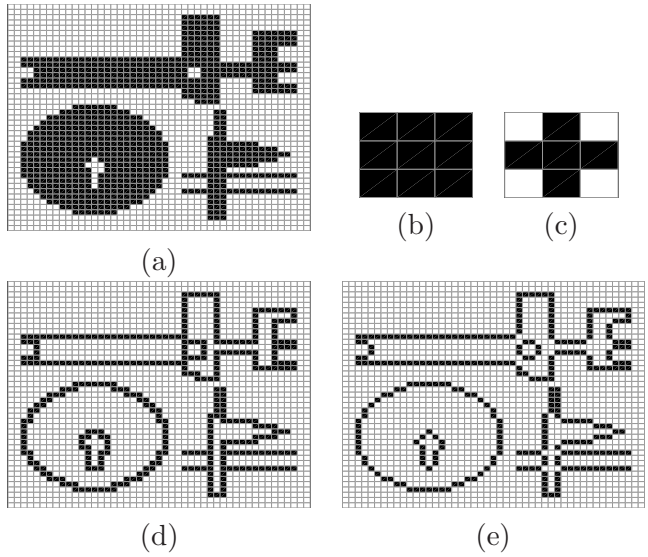
$$\kappa(A) = A - (A \ominus B) \tag{6.14}$$

Powstały w ten sposób kontur jest konturem wewnętrznym, to znaczy takim, że zawiera się w zbiorze pierwotnym A: $\kappa(A) \in A$. Analogicznie możemy zdefiniować kontur zewnętrzny przez:

$$\kappa_z(A) = (A \oplus B) - A \tag{6.15}$$

Wtedy przecięcie zbioru A i jego konturu $\kappa_z(A)$ jest zbiorem pustym.

Szerokość konturu w prosty sposób reguluje wielkość elementu strukturalnego B. Na rysunku 6.5 przedstawiona jest ekstrakcja konturu dla elementu strukturalnego B kwadratowego i krzyżowego o wielkości 3×3. Taki rozmiar SE zawsze daje kontur o szerokości 1-punktowej. element strukturalny o wielkości 5×5 da kontur o szerokości 2-punktowej, o wielkości 7×7 – 3-punktowej, itd.



Rysunek 6.5. Ilustracja wyznaczania konturu obiektu na obrazie binarnym: (a) obraz oryginalny, (b) i (c) element strukturalny, odpowiednio kwadratowy, wielkości 3×3 oraz krzyż 3×3, nie jest zachowana skala pomiędzy obrazem a obiektem strukturalnym, (d) kontur obrazu ekstrakcji elementem strukturalnym (b), (e) kontur obrazu ekstrakcji elementem strukturalnym (c).

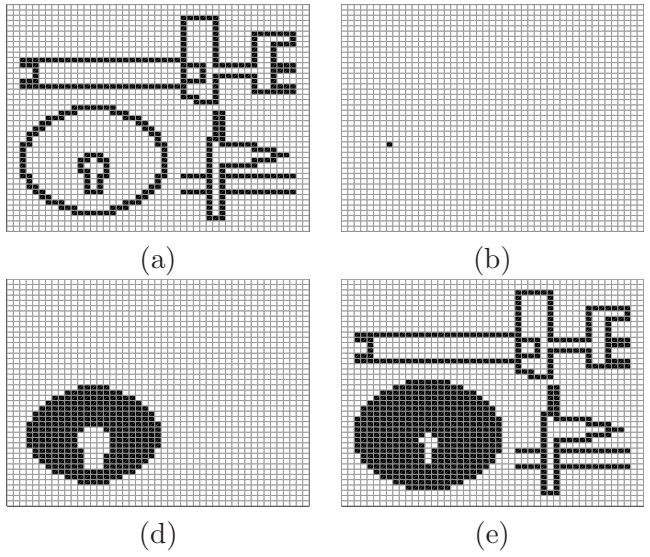
6.2.2. Wypełnianie dziur

Dziura może być zdefiniowana jako obszar tła otoczony przez połączony kontur obiektu. Algorytm wypełnienia tak zdefiniowanej dziury jest iteracyjnym algorytmem opartym na dylatacji i przecięciu. Dla danej dziury

musi istnieć obraz A_h tego samego rozmiaru co obraz oryginalny, wypełniony zerami, z jedną jedynką w dowolnym punkcie w obszarze dziury. Ten punkt początkowy jest niezbędny przy braku zewnętrznej wiedzy na temat obiektu i tła. Iteracyjny proces wypełnienia dziury (ang. *hole filling*) można zdefiniować w następujący sposób:

$$A_h^{(i)} = (A_h^{(i-1)} \oplus B) \cap A^C \qquad i = 1, 2, 3, ... \tag{6.16}$$

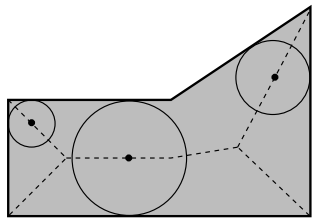
gdzie B jest symetrycznym elementem strukturalnym. Warunkiem zakończenia iteracji w kroku i jest spełnienie warunku $A_h^{(i)} = A_h^{(i-1)}$, to znaczy wtedy gdy cała dziura została już wypełniona. Warunek przecięcia dylatacji z dopełnieniem zbioru A ma za zadanie nie dopuszczać do rozrostu dylatacji poza obszar wypełnianego konturu. Na Rysunku 6.6 przedstawiono wypełnianie jednego z obiektów, rysunek 6.6(b) przedstawia zbiór $A_h^{(0)}$ z punktem początkowym znajdującym się wewnątrz dziury. Algorytm potrzebował 17 iteracji na całkowite wypełnienie wybranego konturu. Użyty element strukturalny był kształtu kwadratowego o rozmiarze 3×3.



Rysunek 6.6. Ilustracja wypełniania dziury wewnątrz konturu obiektu na obrazie binarnym: (a) obraz oryginalny, (b) obraz wypełniania dziury z zaznaczonym punktem startowym wewnątrz dziury $A_h^{(0)}$, (c) obraz w pełni wypełnionej dziury $A_h^{(17)}$, (d) suma obrazu dziury i obrazu pierwotnego $A_h^{(17)} + A$.

6.2.3. Szkieletyzacja

Szkieletyzacja (ang. *skeletonization*) jest procesem, który umożliwia uzyskanie szkieletu obiektu czyli jego punktów osiowych. Jest to technika o olbrzymim znaczeniu praktycznym w przetwarzaniu obrazów medycznych czy rozpoznawaniu pisma. Przez szkielet będziemy rozumieć zbiór wszystkich możliwych środków maksymalnych okręgów, które można wpisać w dany zbiór A. Innymi słowy, jest to taki zbiór punktów, który jest równoodległy od co najmniej dwóch krawędzi zbioru A. W wyniku operacji szkieletyzacji cały obiekt zostaje zredukowany do zbioru linii o szerokości jednego punktu. Przykładowy kształt oraz jego szkielet jest zobrazowany na Rysunku 6.7.



Rysunek 6.7. Przykładowy kształt oraz jego szkielet; kilka maksymalnych okręgów o środkach na szkielecie zostało wpisanych w kształt obiektu.

Przeanalizujmy dwie techniki szkieletyzacji: (1) opartą na erozji i otwarciu oraz (2) opartą na pocienianiu obiektu aż do uzyskania szkieletu.

6.2.3.1. Szkielet erozyjny

Technika szkieletyzacji oparta na operacji erozji i otwarcia tworzy szkielet $S(A)$ obiektu A za pomocą iteracyjnej erozji obiektu A i otwarcia tak powstałego obiektu. Pełny szkielet powstaje poprzez zsumowanie poszczególnych iteracji procesu szkieletyzacji:

$$S(A) = \bigcup_{k=0}^{K} S_k(A) \tag{6.17}$$

gdzie:

$$S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B \tag{6.18}$$

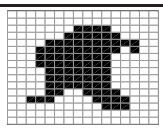
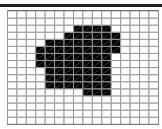
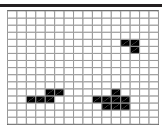
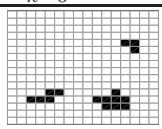
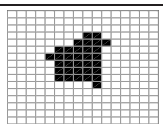
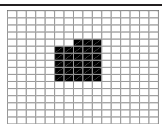
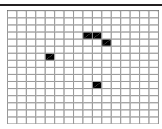
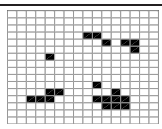
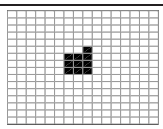
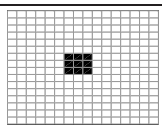
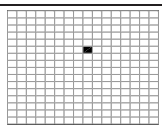
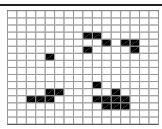
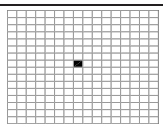
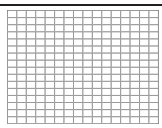
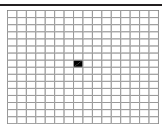
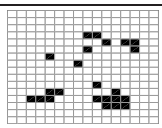
B jest elementem strukturalnym a wyrażenie $(A \ominus kB)$ oznacza k krotną erozję zbioru A:

$$(A \ominus kB) = ((...((A \ominus B) \ominus B) \ominus ...) \ominus B) \tag{6.19}$$

Iteracja ma K cykli, przy czym ostatni cykl to taki, w którym erozja generuje zbiór pusty:

$$K = \max\{k|(A \ominus kB) \neq \phi\} \tag{6.20}$$

Rysunek 6.8 ilustruje ideę szkieletyzacji erozyjnej. Pierwsza kolumna to obraz po k-krotnej erozji (dla $k = 0$ obraz jest oryginalny). W dyskutowanym przykładzie ilość iteracji $K = 4$, piąta iteracja w wyniku erozji wygenerowałaby zbiór pusty. W drugiej kolumnie jest zbiór po operacji otwarcia przeprowadzonej na wyniku erozji zbioru z pierwszej kolumny. Trzecia kolumna przedstawia zbiór będący różnicą zbioru z pierwszej i drugiej kolumny. W czwartej kolumnie są dosumowywane cząstkowe zbiory z trzeciej kolumny. Na dole czwartej kolumny jest wynik szkieletyzacji. Element strukturalny w każdym przypadku jest kwadratem o wielkości 3×3.

k	$A \ominus kB$	$(A \ominus kB) \circ B)$	$S_k(A)$	$\bigcup_{k=0}^{K} S_k(A)$
0				
1				
2				
3				

Rysunek 6.8. Ilustracja szkieletyzacji erozyjnej. Obraz oryginalny jest w pierwszej kolumnie na samej górze. Wynik szkieletyzacji jest na dole ostatniej kolumny. Szczegółowy opis w tekscie.

Metoda szkieletyzacji erozyjnej jest bardzo prostą i szybką techniką sformułowaną w kategoriach erozji i otwarcia ale rezultat nie jest do końca satysfakcjonujący. Po pierwsze jest w wielu miejscach grubszy niż być powinien i po drugie nie ma zapewnionej ciągłości pomiędzy punktami osiowymi. Nieco lepsze rezultaty można uzyskać stosując morfologiczne metody pocieniania.

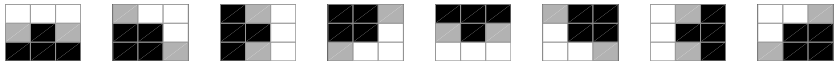
6.2.3.2. Pocienianie

Pocienianie (ang. *thinning*) ma na celu maksymalne "odchudzenie" obiektu tak, żeby jego szerokość w danym miejscu była jednopunktowa.

Pocienianie zbioru A przez element strukturalny B może być zdefiniowane w kategoriach transformacji trafić-chybić (hit-or-miss):

$$A \otimes B = A - (A \circledast B) = A \cap (A \circledast B)^C \tag{6.21}$$

Pocienianie jest również procesem iteracyjnym trwającym tak długo aż różnica zbiorów z dwóch ostatnich iteracji będzie zbiorem pustym. Wykorzystuje się tu konkretny, ustalony element strukturalny razem z jego wersjami obróconymi wokół punktu centralnego. W minimalnej ilości będzie to 4 wersje tego SE obróconych o kąt 0°, 90°, 180° i 270°. Rysunek 6.9 przedstawia element strukturalny w 8 wersjach obróconych o 45° każda.



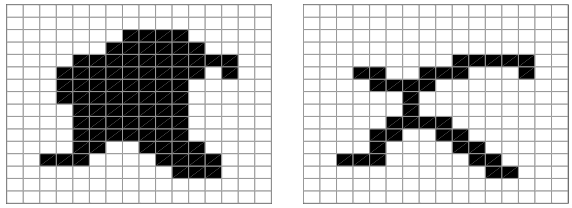
(B_1) (B_2) (B_3) (B_4) (B_5) (B_6) (B_7) (B_8)

Rysunek 6.9. Zbiór elementów strukturalnych używanych do pocieniania obróconych o 45° każdy w stosunku do poprzednika.

Jedna iteracja operacji pocieniania będzie się składała z sekwencji operacji 6.21 zastosowanych jedna na drugiej przy użyciu kolejnych elementów strukturalnych:

$$A \otimes \{B\} = ((...((A \otimes B_1) \otimes B_2)...) \otimes B_n) \tag{6.22}$$

Rysunek 6.10 ilustruje wynik działania pocieniania obrazu binarnego. W tym przypadku potrzebne było 5 iteracji do uzyskania pełnego szkieletu obiektu.



(a) (b)

Rysunek 6.10. Przykład szkieletyzacji opartej na pocienianiu: (a) obraz pierwotny, (b) szkielet obrazu uzyskany przy pomocy operacji pocieniania.

Szkieletyzacja metodą pocieniania przyniosła dużo efektywniejszy rezultat w porównaniu do szkieletyzacji erozyjnej. Jednak i tu często występują niepożądane artefakty w postaci rozgałęzień. Jednym z prostszych rozwiązań problemu gałązek jest użycie algorytmu morfologicznego zwanego obci-

naniem (ang. *pruning*). Jest to również iteracyjny algorytm oparty na transformacji trafił-chybił, dylatacji i przecięciu zbiorów. Pełny opis algorytmu obcinania oraz inne metody szkieletyzacji czytelnik znajdzie w [16, 48, 19].

6.3. Morfologia obrazów skali szarości

Rozszerzenie stosowania metod morfologii matematycznej z obrazów binarnych na obrazy w skali szarości wymaga przejścia z binarnych operacji na funkcje w formie $f(x, y)$ dla obrazu i $b(x, y)$ dla elementu strukturalnego. Obie funkcje są z założenia funkcjami dyskretnymi rzeczywistymi (zazwyczaj o całkowitej precyzji). Element strukturalny w morfologii skali szarości pełni tę samą funkcję co w odpowiedniku binarnym, tzn. jest swego rodzaju wskaźnikiem, które punkty otoczenia należy brać pod uwagę w danej operacji. Przy czym dopuszczalne są dwie formy elementu strukturalnego: płaska oraz nie-płaska. Płaski jest odpowiednikiem binarnego elementu strukturalnego, czyli dopuszcza jedynie dwie wartości: minimalną lub maksymalną. Nie-płaski element strukturalny może być rozumiany w kategoriach obrazu w skali szarości z wartościami wybranymi z odpowiedniego zakresu. Z powodu dużych trudności ze zdefiniowaniem odpowiedniego nie-płaskiego elementu strukturalnego do konkretnej operacji, ten typ SE jest wykorzystywany stosunkowo rzadko. Tak jak w przypadku binarnym, element strukturalny w przypadku skali szarości musi mieć określony punkt centralny, w przeciwnym wypadku zakłada się jego istnienie na przecięciu osi symetrii SE.

W rozdziale 5.2 były już dyskutowane podstawowe operacje morfologiczne dla obrazu w skali szarości: filtr minimum będący odpowiednikiem binarnej erozji i filtr maksimum będący odpowiednikiem binarnej dylatacji oraz operacje otwarcia i zamknięcia. W poniższym rozdziale przedstawionych zostanie zatem kilka podstawowych algorytmów morfologii skali szarości wykorzystujących wspomniane operacje.

6.3.1. Morfologiczne wygładzanie

Ponieważ operacja otwarcia wygasza jasne obiekty mniejsze od elementu strukturalnego a operacja zamknięcia tłumi ciemne detale, często używa się ich kombinacji do wygładzania obrazu i/lub usuwania szumu. Rozważmy operację otwarcia wykonaną na obrazie w skali szarości a następnie zamknięcia na wyniku poprzedniej operacji:

$$f = (f \circ b) \bullet b \tag{6.23}$$

Jak oczekiwano, taka sekwencja usuwa drobne detale obrazu w zależności od kształtu funkcji b. Taka procedura jest często używana w automatycz-

nej analizie obrazu, gdzie operacja otwarcie-zamknięcie jest wykonywana sekwencyjnie na wyniku poprzedniego kroku. To podejście dosyć dobrze wygładza i rozmywa obraz. Przykładowy, lekko zaszumiony obraz w skali szarości jest pokazany na Rysunku 6.11(a), na Rysunkach 6.11(b) i 6.11(c) są jego odszumione i wygładzone wersje. W pierwszym przypadku operacja morfologicznego wygładzania była przeprowadzona z kwadratowym elementem strukturalnym o wielkości 3×3, w drugim z elementem strukturalnym w kształcie dysku o promieniu 4.

6.3.2. Morfologiczny gradient

Gradient morfologiczny może być zdefiniowany za pomocą kombinacji erozji i dylatacji w następujący sposób:

$$f = (f \oplus b) - (f \ominus b) \tag{6.24}$$

Dylatacja pogrubia jasne obszary w obrazie a erozja je pomniejsza. W obu przypadkach jednolite obszary pozostają nienaruszone. Zatem różnica erozji i dylatacji podkreśli przejścia pomiędzy tymi obszarami eliminując jednocześnie obszary jednolite. Powstały w ten sposób obraz ma wzmocnione krawędzie a efekt jest podobny do filtrów krawędziowych. Wynik zastosowania operacji morfologicznego gradientu dla obrazu przedstawionego na Rysunku 6.11(a) jest na Rysunku 6.11(d). Zastosowany element strukturalny miał kształt kwadratu o boku długości 3.

6.3.3. Transformacje top-hat i bottom-hat

Operacja top-hat jest kombinacją obrazu w skali szarości i jego otwarcia zdefiniowaną następująco:

$$f = f - (f \circ b) \tag{6.25}$$

Analogicznie jest zdefiniowana operacja bottom-hat jako kombinacja obrazu i jego zamknięcia:

$$f = (f \bullet b) - f \tag{6.26}$$

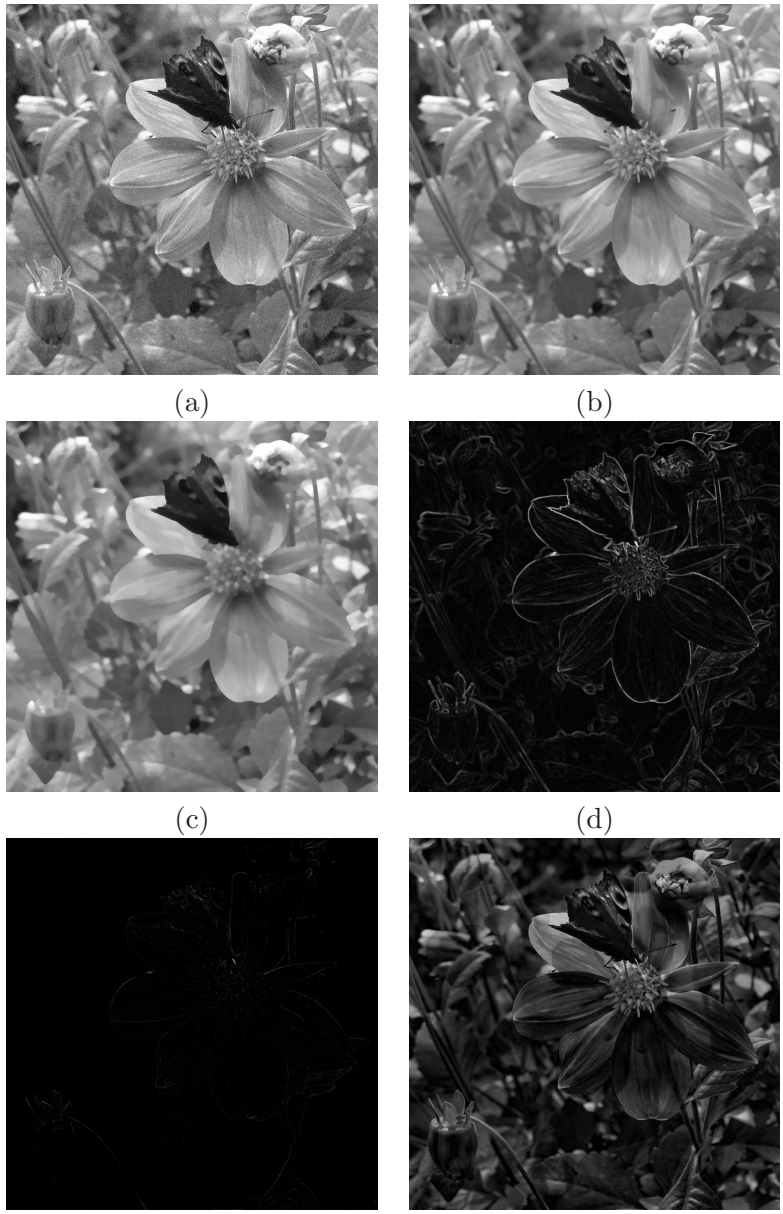
Operacje te mają własności wykrywania ekstremów w obrazie, przy czym transformacja top-hat wyszukuje lokalne maksima a transformacja bottom-hat lokalne minima. Rysunek 6.11(e) przedstawia lokalne ekstrema dla obrazu z Rysunku 6.11(a).

Operacja top-hat wykonywana z elementem strukturalnym o dużym rozmiarze odgrywa ważną rolę w korekcji niejednorodnego oświetlenia w danym obrazie. Taka korekcja jest często zabiegiem wstępnym w bardziej zaawansowanych algorytmach wydobywania pożądanych obiektów z tła lub w procesie segmentacji. Wynik operacji morfologicznej top-hat dla obrazu w skali

szarości z elementem strukturalnym w kształcie dysku o promieniu równym 40 jest przedstawiony na Rysunku 6.11(f).

Zastosowanie operacji morfologicznych zamiast filtracji splotowych często ma swoje uzasadnienie wydajnościowe. Jednakże dużo w tym wypadku zależy od kształtu i wielkości użytego w danej transformacji elementu strukturalnego.

Inne często stosowane algorytmy morfologiczne w przetwarzaniu obrazów w skali szarości, takie jak granulometria czy morfologiczna rekonstrukcja czytelnik znajdzie w pozycjach [16, 39, 21], więcej na temat morfologii matematycznej w ogólności w pozycjach [41, 46, 22].



(a) (b) (c) (d) (e) (f)

Rysunek 6.11. (a) Lekko zaszumiony obraz w skali szarości, (b) obraz odszumiony za pomocą morfologicznego wygładzania z kwadratowym elementem strukturalnym 3×3, (c) obraz wygładzony za pomocą morfologicznego wygładzania z elementem strukturalnym w kształcie dysku o promieniu 4, (d) obraz krawędzi uzyskany za pomocą operacji morfologicznego gradientu z kwadratowym SE o wielkości boku 3×3, (e) lokalne ekstrema obrazu uzyskany za pomocą transformacji top-hat, (f) operacja top-hat z elementem strukturalnym o kształcie dysku o promieniu 40.

Rozdział 7

Przekształcenia w dziedzinie częstotliwości

7.1. Dyskretna Transformata Fouriera

Jednym z istotniejszych narzędzi w przetwarzaniu sygnałów jest transformata Fouriera, która rozkłada dany sygnał na składowe okresowe, sinusowe i kosinusowe. Dane wyjściowe z tej transformacji reprezentują pewną funkcję w dziedzinie Fouriera zwaną częściej częstotliwościową. Taka częstotliwościowa reprezentacja sygnału może być bardzo użyteczna przy manipulacji danymi. Związek pomiędzy funkcją wejściową $f(x)$ a jego transformatą Fouriera jest wyrażany poprzez transformację Fouriera [7]:

$$\mathcal{F}(f(x)) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi\omega x}dx = \Phi(\omega) \tag{7.1}$$

Ten związek przekształca dziedzinę przestrzenną x w dziedzinę częstotliwości ω. Transformata jest odwracalna, a $\mathcal{F}^{-1}$ wyrażone równaniem:

$$\mathcal{F}^{-1}(\Phi(\omega)) = \int_{-\infty}^{\infty} \Phi(\omega)e^{i2\pi\omega x}d\omega = f(x) \tag{7.2}$$

rekonstruuje funkcję $f(x)$ z jego spektrum $\Phi(x)$. W obu przypadkach i jest jednostką urojoną taką, że $i^2 = -1$. Korzystając z wzoru Eulera na reprezentację liczby zespolonej, transformację Fouriera można zapisać równoważnie w następujący sposób:

$$\mathcal{F}(f(x)) = \int_{-\infty}^{\infty} f(x)[\cos(2\pi\omega x) - i\sin(2\pi\omega x)]dx \tag{7.3}$$

Wynika z tego, że nawet jeżeli funkcja $f(x)$ jest funkcją rzeczywistą to jej transformata będzie zespolona. Zwyczajowo, dla celów wizualizacji, używa się modułu liczby zespolonej transformaty (wartość rzeczywista), która w tym przypadku jest nazywana spektrum Fourierowskim lub spektrum częstotliwości.

Transformację Fouriera zdefiniowaną w (7.1) stosuje się w przypadku ciągłych i całkowalnych funkcji. W przypadku przetwarzania obrazów cyfrowych potrzeba jej dyskretnej wersji zwanej Dyskretną Transformatą Fouriera (DFT) rozszerzonej do dwóch wymiarów. DFT definiuje się w następujący sposób:

$$\mathcal{F}(f(x,y)) = \sum_{x=0}^{N-1}\sum_{y=0}^{M-1} f(x,y)e^{-i2\pi(\omega x/N + \nu y/M)} = \Phi(\omega,\nu) \tag{7.4}$$

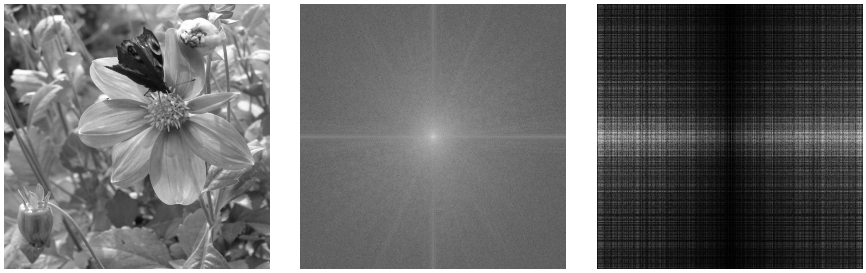
gdzie dwuwymiarowy sygnał $f(x,y)$ (w tym przypadku obraz cyfrowy) jest próbkowany od 0 do $N-1$ w pierwszym i od 0 do $M-1$ w drugim wymiarze. Równanie to może być interpretowane jako: wartość każdego punktu

$\Phi(\omega, \nu)$ jest uzyskiwana przez przemnożenie przestrzennego obrazu przez odpowiednie funkcje bazowe. Funkcjami bazowymi są fale sinusowe i kosinusuwe o rosnącej częstotliwości, to znaczy: $\Phi(0,0)$ reprezentuje część obrazu odpowiadającą średniej jasności, natomiast $\Phi(N-1, M-1)$ reprezentuje najwyższe częstotliwości. Na Rysunku 7.1 jest pokazane spektrum transformaty, z podziałem na moduł liczby zespolonej i jej fazę.

Analogicznie do funkcji ciągłych obraz transformaty może być od-transformowany z powrotem do dziedziny obrazu za pomocą odwrotnej transformaty zdefiniowanej równaniem:

$$\mathcal{F}^{-1}(\Phi(\omega,\nu)) = \sum_{\omega=0}^{N-1}\sum_{\nu=0}^{M-1} \Phi(\omega,\nu) e^{i2\pi(\omega x/N + \nu y/M)} = f(x,y) \tag{7.5}$$

W dziedzinie Fouriera każdy punkt reprezentuje konkretną częstotliwość zawartą w obrazie w dziedzinie przestrzennej. Często odpowiednie dziedziny nazywa się również "przestrzenią obrazu" i "przestrzenią częstotliwości". W przypadku *DFT* transformata nie zawiera wszystkich częstotliwości składających się na obraz a jedynie ich liczbę "wystarczającą" do opisu przestrzennego obrazu. Ta liczba częstotliwości odpowiada liczbie próbek obrazu wejściowego transformacji, to znaczy rozmiar obrazu w dziedzinie przestrzennej i Fouriera jest identyczny.



(a) (b) (c)

Rysunek 7.1. (a) Przykładowy obraz. (b) Wizualizacja transformaty Fouriera dla obrazu: (b) - moduł liczby zespolonej, (c) - faza liczby zespolonej.

Co ważniejsze, bardzo kosztowna obliczeniowo dyskretna transformata Fouriera (w przypadku jednowymiarowym złożoność $O(N^2)$) może być obliczona za pomocą bardzo wydajnej metody zwanej Szybką Transformatą Fouriera (ang. *Fast Fourier Transform - FFT*) o złożoności $O(N \log N)$. Jest to olbrzymie ulepszenie, zwłaszcza w przypadku dużych obrazów.

Transformata Fouriera w procesie przetwarzania obrazów znajduje szeroki wachlarz zastosowań, jak choćby w analizie obrazów, filtracji, rekonstrukcji czy kompresji.

7.2. Dyskretna Transformata Kosinusowa

Dyskretna Transformata Kosinusowa (ang. *Discrete Cosine Transform – DCT*) jest blisko związana z transformatą Fouriera ale używa jedynie liczb rzeczywistych. Formalnie dyskretna transformata kosinusowa jest liniową, odwracalną funkcją $F : \Re^N \mapsto \Re^N$. Istnieje kilka typów tej transformaty różniących się nieznacznie definicją. Najpopularniejszym wariantem jest typ DCT-II zwykle utożsamiany z DCT zdefiniowany jako transformacja:

$$C_k = \sum_{n=0}^{N-1} X_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \tag{7.6}$$

gdzie: C_k są nazywane współczynnikami DCT lub po prostu transformatą, $k = 0, 1, ..., N-1$. Taka transformacja przekształca skończoną liczbę wartości danych X_n w sumę funkcji kosinusowych oscylujących z różnymi częstotliwościami.

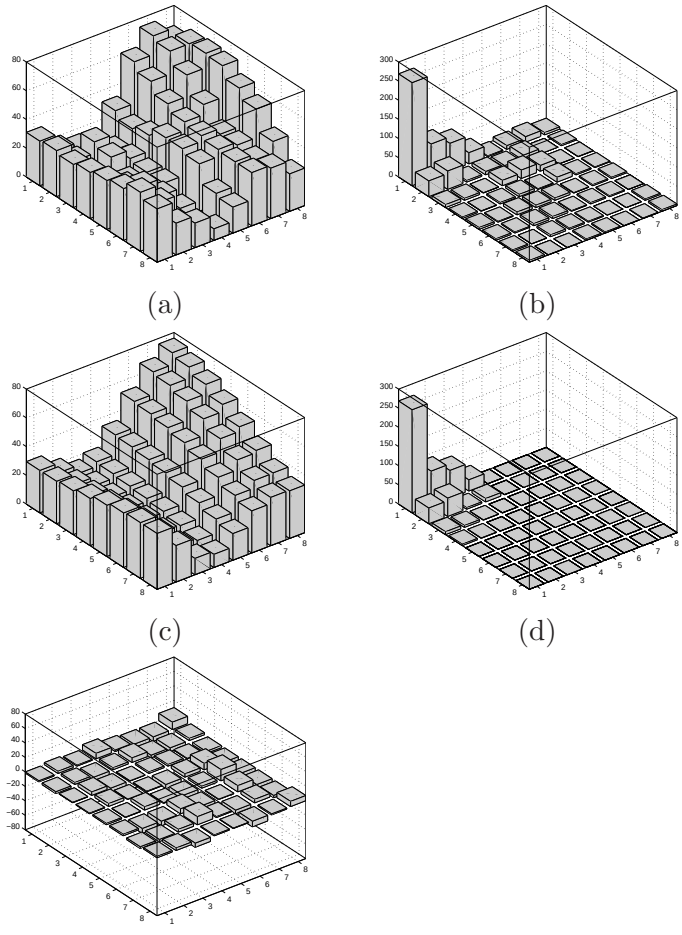
Dla DCT istnieje przekształcenie odwrotne typu DCT-III nazywane zwykle odwrotną transformatą kosinusową (IDCT), które przekształca transformatę z powrotem w sygnał:

$$X_n = \frac{1}{2}X_0 + \sum_{k=1}^{K-1} C_k \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \tag{7.7}$$

Przekształcenie wielowymiarowe DCT jest separowalne co implikuje, że dowolnie-wymiarową transformatę można uzyskać przez kolejne wykonanie jednowymiarowych przekształceń we wszystkich wymiarach. Na przykład, dla obrazu cyfrowego (sygnału dwuwymiarowego) sprowadza się do obliczenia transformaty DCT dla wszystkich wierszy danego obrazu, a następnie przekształcenie tych współczynników kolejnym zestawem operacji DCT liczonych po wszystkich kolumnach. Kolejność wyboru wymiarów jest dowolna.

Niezwykle istotną cechą transformaty kosinusowej, z punktu widzenia przetwarzania sygnałów cyfrowych jest jej niewielka wrażliwość na destrukcję wartości o wysokich częstotliwościach. Większość informacji o sygnale jest bowiem zgromadzona w kilku komponentach DCT o niskiej częstotliwości. Co więcej wartość DCT o najniższej częstotliwości odpowiada wartości średniej transformowanych danych. Ilustruje to przykład na Rysunku 7.2.

Na Rysunku 7.2(a) jest przedstawiony dwuwymiarowy sygnał zawierający 64 próbki w rozdzielczości 8×8. Sygnał ten został następnie poddany transformacji kosinusowej, której wynik jest (co do wartości bezwzględnej) pokazany na Rysunku 7.2(b). Wyraźnie widać informację gromadzącą się w komponentach transformaty o niskiej częstotliwości. Komponenty o większych współrzędnych odzwierciedlają wyższe częstotliwości przestrzenne,



Rysunek 7.2. Ilustracja własności transformaty kosinusowej. Rysunek (a) – dwuwymiarowy zbiór danych o wymiarze 8×8, (b) – współczynniki transformaty kosinusowej zbioru (a), (d) – współczynniki transformaty po częściowym wyzerowaniu wartości, (c) – zbiór danych po odwrotnej transformacji kosinusowej zbioru (d), (e) – różnica wartości pomiędzy zbiorami (a) i (c).

które reprezentują szybsze zmiany wartości w danej próbce. Zauważamy, że im dalej współczynniki oddalone są od składowej (1,1), tym posiadają niższe i bardziej zbliżone do siebie wartości.
Na Rysunku 7.2(d) większość elementów transformaty została wyzerowana – dokładnie 75% komponentów, co oznacza 75 procentową redukcję ilości informacji w próbce. Tak zmodyfikowana transformata została za pomocą odwrotnej transformacji kosinusowej przeliczona z dziedziny częstotliwości

z powrotem do dziedziny danych (Rysunek 7.2(c)). Pomimo dosyć znacznej destrukcji sygnału w transformacie widać wyraźną korelację pomiędzy wartościami danych z przed transformaty (a) i po transformacie odwrotnej (c). Rysunek 7.2(e) przedstawia różnicę wartości sygnału oryginalnego i przekształconego. W wyniku powyższej transformacji sygnał uległ około 15% degradacji za cenę zmniejszenia ilości informacji do 25% pierwotnej wielkości.

Złożoność obliczeniowa Dyskretnej Transformaty Kosinusowej wynosi podobnie jak transformaty Fouriera $O(N^2)$ ale dzięki dzięki algorytmom podobnym do FFT złożoność ta może być obniżona do $O(NlogN)$.

7.3. Uwagi implementacyjne

Samodzielna implementacja algorytmów liczenia FFT jest procesem dosyć skomplikowanym i wykraczającym poza ramy niniejszej pozycji. Bardzo dobre studium szybkiej transformaty Fouriera znajdzie czytelnik w [35, 4] Dalsze rozważania będą oparte na istniejącej wieloplatformowej implementacji zwanej FFTW [15]. Dostarczana w postaci biblioteki, napisana w języku C, implementacja FFTW potrafi obliczać DFT oraz DCT dowolnie-wymiarowe, o dowolnej wielkości dla danych rzeczywistych jak i zespolonych. Biblioteka ta jest wolnodostępna na licencji GPL.

7.3.1. C/C++

Dla programów pisanych w języku C++ biblioteki FFTW są dostarczane w postaci natywnych bibliotek systemowych na konkretną platformę. Łączenie bibliotek FFTW z bibliotekami Qt wymaga dodania do pliku projektu następujących wpisów:

```
LIBS = -L/sciezka/katalogu/biblioteki -lfftw3
INCLUDEPATH = -I/sciezka/katalogu/pliku/naglowkowego
```

W przypadku dołączania konkretnej wersji biblioteki można wywołanie `-lfftw3` zastąpić konkretną wersją poprzez wywołanie `-l:libfftw3.so.x.y.z`. Następnie, w celu odświeżenia pliku reguł makefile, należy uruchomić program `qmake` (w przypadku QtCreatora w menu wybrać “Build/Run qmake“) i przebudować projekt (”Build/Rebuild Project“).
W pliku, w którym są używane funkcje z biblioteki FFTW należy dołączyć plik `fftw3.h`
Plik biblioteki `libfftw3.dll` – w przypadku Windows lub `libfftw3.so` – w przypadku Linuxa, musi być „widoczny” dla pliku wykonywalnego, tzn. musi

być albo w katalogu, w którym system trzyma biblioteki albo w tym samym katalogu co plik wykonywalny. Dla systemu Linux najlepszym rozwiązaniem będzie wykorzystanie biblioteki FFTW z używanej dystrybucji.

Samo użycie funkcji biblioteki do obliczenia **Szybkiej Transformaty Fouriera (FFT)** ilustruje listing 7.1.

Listing 7.1. Użycie funkcji biblioteki FFTW.

```
#include <fftw3.h>
...
{
  int W, H;
  QImage image(W, H, QImage::Format_RGB32);

  fftw_complex *in, *out;
  in  = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)
                                                  *W*H);
  out = (fftw_complex*)fftw_malloc(sizeof(fftw_complex)
                                                  *W*H);

  QRgb* p = (QRgb*)image.bits();
  for(int i = 0; i < W*H; i++)
  {
    in[i][0] = qRed(p[i]);
    in[i][1] = 0;
  }

  fftw_plan p;
  p = fftw_plan_dft_2d(H, W, in, out, FFTW_FORWARD,
                                      FFTW_ESTIMATE);
  fftw_execute(p);
  fftw_destroy_plan(p);

    //   ...   operacje   na   transformacie   ...

  p = fftw_plan_dft_2d(H, W, out, in, FFTW_BACKWARD,
                                      FFTW_ESTIMATE);
  fftw_execute(p);
  fftw_destroy_plan(p);

  p = (QRgb*)image.bits();
  for(int i = 0; i < W*H; i++)
  {
    p[i] = qRgb( in[i][0], qGreen(p[i]), qBlue(p[i]) );
  }

  fftw_free(in); fftw_free(out);
}
```

Linia 1 – dołączony plik nagłówkowy z deklaracjami struktur i funkcji biblioteki FFTW.

Linie 4-5 – tworzony jest obraz `image` typu `QImage` o rozmiarze $W \times H$ i 32-bitowej głębi bitowej.

Linie 7-10 – definicja tablic liczb zespolonych, które będą przechowywały dane transformaty. Strukturę `fftw_complex` można traktować jak tablicę o dwóch elementach typu **`double`**:

```
1 typedef double fftw_complex[2];
```

W elemencie `fftw_complex[0]` jest przechowywana wartość części rzeczywistej liczby zespolonej, w elemencie `fftw_complex[1]` wartość części urojonej liczby zespolonej. Do alokacji pamięci została użyta biblioteczna funkcja:

```
1 void *fftw_malloc(size_t n);
```

dbająca o poprawną alokację i odpowiednie wyrównanie pamięci.
Tablica `in` oraz tablica `out` mają wielkość obrazu ($W \times H$) razy wielkość zmiennej typu `fftw_complex`. Zmienna `in` będzie przechowywała wartości z dziedziny obrazu, czyli wartości przed transformatą, natomiast zmienna `out` wartości z dziedziny częstotliwości, tzn. wartości transformaty.

Linie 13-18 – dane z obrazu `image` są kopiowane do zmiennej `in`. Wybieramy tutaj tylko składową czerwoną obrazu, i jej wartości kopiujemy do części rzeczywistej tablicy `in`, część urojona zostaje wyzerowana.

Linie 20-21 – za pomocą funkcji `fftw_plan_dft_2d` tworzony jest plan transformaty,czyli struktura, która zawiera wszystkie elementy niezbędne do przeprowadzenia obliczeń FFT:

```
fftw_plan fftw_plan_dft_2d(int n0, int n1,
          fftw_complex *in, fftw_complex *out,
          int sign, unsigned flags);
```

W omawianym przykładzie tworzony jest plan do obliczeń dwu-wymiarowej pełnej transformaty Fouriera o rozdzielczości $H \times W$. Tablica `in` jest przekazywana jako tablica wejściowa obliczeń. Wynik obliczeń zostanie zapisany do tablicy `out`. Wartość parametru `sign`= `FFTW_FORWARD`(wartość tej zmiennej to -1) instruuje plan, że jest to transformata wprost. Flaga `FFTW_ESTIMATE` określa sposób obliczeń transformaty.

Linia 23 – wykonuje obliczenia transformaty zdefiniowane w planie `p` za pomocą funkcji:

```
1 void fftw_execute(const fftw_plan plan);
```

Linia 24 – usuwa z pamięci niepotrzebny już plan `p` za pomocą funkcji:

```
1 void fftw_destroy_plan(fftw_plan plan);
```

Linia 26 – symbolicznie zaznaczone są odpowiednie obliczenia/przekształcenia samej transformaty.

Linia 27 – zdefiniowany zostaje nowy plan o identycznym rozmiarze jak poprzedni ale zamienione są miejscami tablice wejściowe i wyjściowe transformaty, a sama transformata będzie wykonana 'w tył', zatem będzie to transformata odwrotna. Decyduje o tym znak przekazany za pomocą wartości `FFTW_BACKWARD = +1`. Linia 30 wykona obliczenia dla tego planu wykonując transformatę odwrotną i w linii 31 plan zostaje usunięty z pamięci.

Linie 33-37 – kopiowanie części rzeczywistej wyniku transformaty odwrotnej z powrotem do kanału czerwonego obrazu `image`.

Linia 39 – zwolnienie pamięci używanej do przechowywania tablic transformaty `in` i `out` za pomocą funkcji:

```
1 void fftw_free(void *p);
```

Istnieje również wersja transformaty typu "real to complex" umożliwiająca obliczenie transformaty tylko dla elementów rzeczywistych. Jedyną różnicą w powyższym kodzie będzie stworzenie odpowiedniego planu za pomocą funkcji:

```
fftw_plan fftw_plan_dft_r2c_2d(int n0, int n1, double *in,
                        fftw_complex *out, unsigned flags);
```

Dodatkowo dane wejściowe są przechowywane w tablicy z wartościami typu `double`. Parametr `flags` może mieć wartości identyczne z omawianą powyżej funkcją `fftw_plan_dft_2d()`. Nie ma tu natomiast parametru określającego kierunek obliczeń – powyższa funkcja zawsze liczy transformatę w przód. Dla transformaty odwrotnej istnieje osobna funkcja:

```
fftw_plan fftw_plan_dft_c2r_2d(int n0, int n1,
          fftw_complex *in, double *out, unsigned flags);
```

W tym przypadku dane wejściowe (transformata) są typu `fftw_complex` a dane wyjściowe typu `double`. Pewną niedogodnością tej funkcji jest fakt, że niszczy ona dane przechowywane w tablicy wejściowej.

Proszę zauważyć, że przeprowadzone tu operację dotyczą tylko jednej składowej obrazu RGB – czerwonej. Dla przeprowadzenia obliczeń wykorzy-

stujących transformatę Fouriera na pełnym obrazie RGB należy powyższe obliczenia przeprowadzić trzykrotnie – dla każdej składowej.

Do obliczenia dwuwymiarowej **Dyskretnej Transformaty Kosinusowej (DCT)** wykorzystamy funkcję biblioteki FFTW:

```
fftw_plan fftw_plan_r2r_2d(int n0, int n1,
                           double *in, double *out,
                           fftw_r2r_kind kind0,
                           fftw_r2r_kind kind1,
                           unsigned flags);
```

tworzącą plan transformaty typu *real-to-real*. Funkcja ta tworzy plan liczący dwuwymiarową transformatę tablicy `in` o wymiarach $n0 \times n1$. Wynik zostanie zapisany do tablicy `out`. Tablice, zarówno wejściowa `in` jak i wyjściowa `out` są jednowymiarowymi tablicami przechowującymi informacje o dwuwymiarowych danych kolejno wierszami. Zmienne `kind0` oraz `kind1` określają typ transformaty. Dla transformaty kosinusowej (DCT) oba parametry powinny mieć wartość `FFTW_REDFT10`, a dla odwrotnej transformaty kosinusowej (IDCT) wartość `FFTW_REDFT01`. Parametr `flags` ma identyczne znaczenie jak w przypadku pełnej transformaty Fouriera i sugerowaną jego wartością jest `FFTW_ESTIMATE`.

Listing 7.2 ilustruje użycie funkcji `fftw_plan_r2r_2d()` do obliczenia DCT dla tablicy 8×8 oraz transformaty odwrotnej IDCT dla tego samego bloku danych.

Listing 7.2. Użycie funkcji biblioteki FFTW do obliczenia DCT.

```
1  #include <fftw3.h>
2  #include <algorithm>
3  #include <cstdlib>
4  ...
5  {
6    double *in, *out;
7    in  = (double*)fftw_malloc(sizeof(double)*64);
8    out = (double*)fftw_malloc(sizeof(double)*64);
9
10   std::generate_n(in, 64, std::rand);
11
12   fftw_plan p = fftw_plan_r2r_2d(8, 8, in, out,
13                  FFTW_REDFT10, FFTW_REDFT10, FFTW_ESTIMATE);
14
15   fftw_execute(p);
16   fftw_destroy_plan(p);
17
18     //   ...   operacje   na   transformacie   out   ...
19
```

```
  p = fftw_plan_r2r_2d(8, 8, out, in,
      FFTW_REDFT01, FFTW_REDFT01, FFTW_ESTIMATE);

  fftw_execute(p);
  fftw_destroy_plan(p);
  fftw_free(in); fftw_free(out);
}
```

7.3.2. Java

Dla języka Java zostały stworzone odpowiednie nakładki (ang. *wrappery*) na bibliotekę FFTW umożliwiające korzystanie z jej funkcjonalności. Jednym z popularniejszych jest projekt nazwany **jfftw3** dostępny na stronie *https://jfftw3.dev.java.net/*. Aktualna wersja tworzy interfejsy javowe do funkcji biblioteki FFTW w wersji 3 zarówno na platformę MS Windows jaki i Linuxa. Sama nakładka składa się z 3 plików: (1) biblioteki `jfftw3.dll` dla Windows lub `libjfftw3.so` w przypadku Linuxa, (2) biblioteki javowej `jfftw3.jar` zawierającej odpowiednie interfejsy oraz (3) biblioteki `gluegen-re.jar` umożliwiającej wykonywanie wywołań bibliotek C. Odpowiednie wywołania Javy są jedynie nakładkami na wywołania biblioteki FFTW a ich nazwy są poprzedzone literą `j`. Dla pełniejszego obrazu zalecamy zapoznanie się z informacjami zawartymi w poprzednim podrozdziale dotyczącym użycia biblioteki FFTW w programie pisanym w języku C/C++. Podczas uruchamiania programu niezbędne jest wskazanie wirtualnej maszynie Javy katalogu z bibliotekami za pomocą parametru `-Djava.library.path=/sciezka/biblioteki`:

 # java -Djava.library.path=/sciezka/biblioteki -jar Program.jar

Na listingu 7.3 pokazane jest przykładowe użycie FFTW w programie Javy.

Listing 7.3. Obliczanie Szybkiej Transformaty Fouriera w języku Java przy użyciu funkcji biblioteki FFTW.

```
import static com.schwebke.jfftw3.JFFTW3.*;
import java.nio.DoubleBuffer;
...
{
  ...
  int W;
  int H;
  long in = jfftw_complex_malloc(W*H);
  long out = jfftw_complex_malloc(W*H);

  long plan = jfftw_plan_dft_2d(H, W, in, out,
```

```
12                JFFTW_FORWARD, JFFTW_ESTIMATE);
13
14    DoubleBuffer inb = jfftw_complex_get(in);
15    DoubleBuffer outb = jfftw_complex_get(out);
16
17    for(int i=0; i<W*H; i++)
18    {
19      inb.put(2*i, image_data[i]);
20      inb.put(2*i+1, 0);
21    }
22
23    jfftw_execute(plan);
24
25      //      ...    operacje    na    transformacie    ...
26
27    jfftw_destroy_plan(plan);
28    jfftw_complex_free(in);
29    jfftw_complex_free(out);
30  }
```

Linie 1-2 – odpowiednie importy bibliotek dla Javy.

Linie 8-9 – za pomocą funkcji `jfftw_complex_malloc(int)` alokowana jest pamięć na odpowiednie tablice liczb zespolonych. Funkcja ta zwraca w wyniku uchwyt typu `long` jednoznacznie identyfikujący zaalokowaną tablicę.

Linia 11 – tworzony jest plan transformaty przy pomocy funkcji `jfftw_plan_dft_2d()`, która również zwraca uchwyt typu `long` do odpowiedniego planu. W powyższym przykładzie będzie to dwuwymiarowa transformata Fouriera w przód o rozmiarze $H \times W$, liczona dla danych wejściowych zawartych w tablicy `in`. Wynik transformacji zostanie zapisany w tablicy `out`. Warto w tym miejscu wspomnieć, że pomimo dwuwymiarowego charakteru transformaty wszystkie obliczenia odbywają się na jednowymiarowych, liniowych tablicach, w których kolejne wiersze obrazu są zapisywane bezpośrednio po poprzedzających je.

Linie 14-15 – ilustrują sposób dostępu do zawartości tablic `in` i `out` alokowanych w liniach 8 i 9. Funkcja `jfftw_complex_get(long v)` zwraca obiekt typu `DoubleBuffer` dla konkretnego uchwytu `v` identyfikującego żądaną tablicę.

Linie 17-21 – tablica `in` za pomocą obiektu buforowego `inb` wypełniana jest danymi obrazu. Liczby zespolone nie są tu reprezentowane przez osobne struktury a zapisywane w tablicy kolejno, tak że część rzeczywista jest na parzystym indeksie a część urojona na nieparzystym indeksie tablicy. Stąd też tablice `inb` i `outb` mają długość dwa razy większą niż sugerowałaby to wielkość obrazu. Analogicznie jak to było dla języka C/C++ część rzeczywista jest uzupełniana danymi obrazu a część urojona zerowana.

Linia 23 – wywołanie funkcji `jfftw_execute(long)`, która w parametrze przyjmuje uchwyt do planu transformaty i wykonuje transformację Fouriera na danych tego planu.
W dalszej kolejności następują dowolne przekształcenia tablicy zawierającej transformatę Fouriera `outb`.

Linia 27 – usunięcie planu za pomocą funkcji `jfftw_destroy_plan(long)`, przyjmującej w parametrze uchwyt do niszczonego planu.

Linie 28-29 – dealokacja tablic transformaty `in` i `out` za pomocą funkcji `jfftw_complex_free(long)` przyjmującej w parametrze uchwyt do obszaru pamięci, który ma zwolnić.

Dyskretną Transformatę Kosinusową (DCT) można obliczyć wykorzystując biblioteczną funkcję:

```
public static native
long jfftw_plan_r2r_2d(int n0, int n1,
                       long in, long out,
                       int kind0, int kind1, int flags
                       );
```

tworzącą plan transformaty typu *real-to-real*. Funkcja ta tworzy plan liczący dwuwymiarową transformatę tablicy `in` o wymiarach $n0 \times n1$. Wynik zostanie zapisany do tablicy `out`. Tablice, zarówno wejściowa `in` jak i wyjściowa `out` są jednowymiarowymi tablicami przechowującymi informacje o dwuwymiarowych danych kolejno wierszami. Zmienne `kind0` oraz `kind1` określają typ transformaty. Dla transformaty kosinusowej (DCT) oba parametry powinny mieć wartość `FFTW_REDFT10`, a dla odwrotnej transformaty kosinusowej (IDCT) wartość `FFTW_REDFT01`. Parametr `flags` ma identyczne znaczenie jak w przypadku pełnej transformaty Fouriera i sugerowaną jego wartością jest `FFTW_ESTIMATE`.

Listing 7.4 ilustruje użycie funkcji `jfftw_plan_r2r_2d()` do obliczenia DCT dla dwuwymiarowej tablicy o wielkości 8×8.

Listing 7.4. Obliczanie Dyskretnej Transformaty Kosinusowej w języku Java przy użyciu funkcji biblioteki FFTW.

```
1 import static com.schwebke.jfftw3.JFFTW3.*;
2 import java.nio.DoubleBuffer;
3 ...
4 {
5   ...
6   long in = jfftw_real_malloc(8*8);
7   long out = jfftw_real_malloc(8*8);
8
9   long plan = jfftw_plan_r2r_2d(8, 8, in, out,
```

```
10                JFFTW_REDFT10, JFFTW_REDFT10,
11                JFFTW_ESTIMATE);
12
13    DoubleBuffer inb = jfftw_real_get(in);
14    DoubleBuffer outb = jfftw_real_get(out);
15
16    for(int i=0; i<8*8; i++)
17      inb.put(i, i);
18
19    jfftw_execute(plan);
20
21    for(int i=0;i<8*8;i++)
22    {
23      System.out.print(outb.get(i)+" ");
24    }
25
26    jfftw_destroy_plan(plan);
27    jfftw_complex_free(in);
28    jfftw_complex_free(out);
29  }
```

Istotną różnicą w stosunku do obliczeń dla transformaty Fouriera jest użycie innej funkcji tworzącej plan transformaty, tj. `jfftw_plan_r2r_2d()` w linii 9. Dodatkowo, w tym przypadku wszelkie obliczenia są przeprowadzane na liczbach rzeczywistych typu **`double`**. W liniach 21-24 wynik transformaty jest wyświetlany na ekran.

7.3.3. C#

Język C# potrafi bezpośrednio wywoływać funkcje zawarte w bibliotekach `dll` przy pomocy importera `[DLLImport()]`, jednakże byłoby to dosyć uciążliwe i pracochłonne. Wygodniej będzie skorzystać z odpowiedniej nakładki (ang. *wrapper*) na bibliotekę FFTW. Zalecana przez twórców FFTW nakładka została stworzona przez Tamasa Szalay i jest dostępna pod adresem *http://www.sdss.jhu.edu/∼tamas/bytes/fftwcsharp.html*. Nakładka wykorzystuje oryginalną bibliotekę FFTW w wersji 3 przy pomocy pomocniczej, opakowującej biblioteki `fftwlib.dll`. Obie biblioteki muszą być widoczne dla uruchamianego programu. Dla pełniejszego obrazu zalecamy zapoznanie się z informacjami zawartymi w poprzednim podrozdziale dotyczącym użycia biblioteki FFTW w programie pisanym w języku C/C++. Na listingu 7.5 pokazane jest przykładowe obliczenie Szybkiej Transformaty Fouriera (FFT) dla obrazu typu `Bitmap`.

Listing 7.5. Użycie funkcji biblioteki FFTW w języku C#.

```
1 using System.Runtime.InteropServices;
2 using fftwlib;
```

```
3 ...
4 {
5   Bitmap image;
6   IntPtr pin, pout;
7   float[] fin, fout
8
9   pin = fftwf.malloc(W*H*8);
10  pout = fftwf.malloc(W*H*8);
11  fin = new float[W*H*2];
12  fout = new float[W*H*2];
13
14  BitmapData bdata = image.LockBits(new Rectangle(0,0,W,H),
15      ImageLockMode.ReadWrite, PixelFormat.Format32bppRgb);
16
17  try{
18    unsafe {
19      IntPtr Scan0 = bdata.Scan0;
20      int* p = (int*)Scan0.ToPointer();
21      for(int y=0; y<image.Height; ++y) {
22        int ysh = y*image.Width;
23        for(int x=0; x<image.Width; ++x) {
24          fin[2*(ysh+x)] = csred(p[ysh+x]);
25          fin[2*(ysh+x)+1] = 0;
26        }
27      }
28    }
29  } finally {
30    image.UnlockBits(bdata);
31  }
32
33  Marshal.Copy(fin, 0, pin, W*H*2);
34
35  IntPtr plan = fftwf.dft_2d(H, W, pin, pout,
36         fftw_direction.Forward, fftw_flags.Estimate);
37  fftwf.execute(plan);
38
39  Marshal.Copy(pout, fout, 0, W*H*2);
40
41    //      ...   operacje   na   transformacie   ...
42
43  fftwf.free(pin);
44  fftwf.free(pout);
45  fftwf.destroy_plan(plan);
46 }
```

Linie 1-2 – zawierają niezbędne importy z punktu widzenia biblioteki FFTW. Sama biblioteka `fftwlib.dll` musi być również dodana do projektu jako referencja.

Linie 9-10 – alokowane jest miejsce na tablice transformaty. Użyta tu funk-

cja `fftw.malloc(System.Int32 size)` alokuje pamięć zoptymalizowaną pod bibliotekę FFTW ale nie zarządzalną przez C#. Parametr `size` określa ile pamięci w bajtach zaalokować. W powyższym przykładzie potrzebna ilość pamięci to wielkość obrazu (W*H) × wielkość typu float (4) × wielkość liczby zespolonej (2).

Linie 11-12 – na tak zaalokowanych tablicach nie można jednak operować wprost. Stąd alokacja pomocniczych tablic typu `float` dla liczb zespolonych. Same liczby zespolone są zapisywane w tablicy kolejno: część rzeczywista na parzystym indeksie i część urojona na nieparzystym indeksie tablicy.

Linie 14-31 – kopiowanie zawartości kanału czerwonego obrazu `image` do tablicy `fin`. W linii 24 została użyta funkcja `csred(int)` do uzyskania wartości czerwieni z całego koloru zdefiniowana na Listingu 1.11.

Linia 33 – kopiowanie za pomocą metod statycznych klasy `Marshal` zawartości zarządzalnej tablicy `fin` do niezarządzalnej tablicy `pin`.

Linia 35 – definiowany jest plan dwuwymiarowej transformaty w przód i w 37 linii transformata jest obliczana.

Linia 39 – kopiowanie bloku pamięci niezarządzalnej `pout` do zarządzalnej `fout`. W tym momencie można dokonywać operacji na transformacie zapisanej w tablicy `fout`.

Linie 43-44 – dealokacja pamięci zajmowanej przez tablice `pin` i `pout`.

Linia 45 – niszczony jest plan `plan` transformaty.

Dyskretną Transformatę Kosinusową (DCT) można obliczyć wykorzystując statyczną metodę klasy `fftwf`:

```
public static
IntPtr r2r_2d(int n0, int n1,
              IntPtr in, IntPtr out,
              fftw_kind kind0, fftw_kind kind1,
              fft_flags flags);
```

tworzącą plan transformaty typu *real-to-real*. Funkcja ta tworzy plan liczący dwuwymiarową transformatę tablicy `in` o wymiarach $n0 \times n1$. Wynik zostanie zapisany do tablicy `out`. Tablice, zarówno wejściowa `in` jak i wyjściowa `out` są jednowymiarowymi tablicami przechowującymi informacje o dwuwymiarowych danych kolejno wierszami. Zmienne `kind0` oraz `kind1` określają typ transformaty. Dla transformaty kosinusowej (DCT) oba parametry powinny mieć wartość `FFTW_REDFT10`, a dla odwrotnej transformaty kosinusowej (IDCT) wartość `FFTW_REDFT01`. Parametr `flags` ma identyczne znaczenie jak w przypadku pełnej transformaty Fouriera i sugerowaną jego wartością jest `FFTW_ESTIMATE`.

Listing 7.6 ilustruje użycie funkcji `r2r_2d()` do obliczenia DCT dla dwuwymiarowej tablicy o wielkości 8×8.

Listing 7.6. Obliczanie Dyskretnej Transformaty Kosinusowej w języku C# przy użyciu funkcji biblioteki FFTW.

```
1  using System.Runtime.InteropServices;
2  using fftwlib;
3  ...
4  {
5    IntPtr pin, pout;
6    float[] fin, fout
7
8    pin = fftwf.malloc(8*8*4);
9    pout = fftwf.malloc(8*8*4);
10   fin = new float[8*8];
11   fout = new float[8*8];
12
13   for(int y=0; y<8*8; ++y) {
14     fin[i] = i;
15   }
16
17   Marshal.Copy(fin, 0, pin, 8*8);
18
19   IntPtr plan = fftwf.r2r_2d(8, 8, pin, pout,
20                  fftw_kind.REDFT10, fftw_kind.REDFT10,
21                  fftw_flags.Estimate);
22   fftwf.execute(plan);
23
24   Marshal.Copy(pout, fout, 0, 8*8);
25
26     //      ...    operacje    na    transformacie    ...
27
28   fftwf.free(pin);
29   fftwf.free(pout);
30   fftwf.destroy_plan(plan);
31 }
```

Istotną różnicą w stosunku do obliczeń dla transformaty Fouriera jest użycie innej metody statycznej klasy `fftwf` tworzącej plan transformaty, tj. `r2r_2d()` w linii 19. Dodatkowo, w tym przypadku wszelkie obliczenia są przeprowadzane na liczbach rzeczywistych typu `float`.

7.3.4. Matlab

Środowisko Matlaba dysponuje szeregiem funkcji umożliwiających obliczanie transformaty Fouriera oraz transformaty kosinusowej.

Dla dwuwymiarowego przypadku obliczenie **Dyskretnej Transformaty Fouriera (FFT)** realizuje funkcja:

```
Y = fft2(X)
```

gdzie `X` jest macierzą liczb typu `single` lub **`double`**. Wynik transformaty jest umieszczany w macierzy `Y`. Odwrotną transformatę Fouriera wyznacza funkcja:

```
Y = ifft2(X)
```

o identycznych parametrach jak w przypadku funkcji `fft2()`. Listing 7.7 ilustruje sposób obliczenia transformaty Fouriera dla kanału czerwonego obrazu RGB.

Listing 7.7. Obliczanie transformaty fouriera dla kanału czerwonego obrazu RGB.

```
1 image = imread('filename');
2 fft_image = fft2(image(:,:,1));
3 imshow(log(abs(fftshift(fft_image))),[]),
4        colormap(jet(64)), colorbar
```

Warte zauważenia jest, że środowisko Matlab wykorzystuje do obliczeń transformat bibliotekę FFTW. Do kontrolowania działania tej biblioteki służy funkcja:

```
fftw('planner', method)
```

umożliwiająca wybór sposobu optymalizacji obliczeń FFT. Wartości parametru `method` definiuje sama biblioteka FFTW.

Zdefiniowana jest również funkcja:

```
Y = fftshift(X)
```

przestawiająca ćwiartki transformaty zgodnie z Rysunkiem 7.3.

Dyskretną Transformatę Kosinusową (DCT) w wersji dwuwymiarowej realizuje funkcja:

```
Y = dct2(X)
```

zdefiniowana w pakiecie Image Processing Toolbox. Funkcja ta zwraca w wyniku macierz `Y` będącą dwuwymiarową transformatą kosinusową macierzy `X`. Odwrotną transformatę kosinusową oblicza funkcja:

```
1 Y = idct2(X)
```

7.4. Wizualizacja transformaty Fouriera

Ponieważ transformata Fouriera jest reprezentowana przez liczby zespolone jej wizualizacja na płaszczyźnie musi być pewnym jej rzutem lub uproszczeniem. Przyjmując, że F_I jest transformatą obrazu I, wizualizować można jej część rzeczywistą lub urojoną:

a) $I_r = realis(F_I)$
b) $I_r = imaginalis(F_I)$

lub spektrum Fouriera i fazę:

a) $I_r = |F_I| \mapsto |z| = \sqrt{\Re^2 + \Im^2}$
b) $I_r = \phi = \arctan\left(\frac{\Im}{\Re}\right)$

W przypadku wyboru części rzeczywistej lub urojonej można dodatkowo obliczyć wartość bezwzględną każdej liczby:

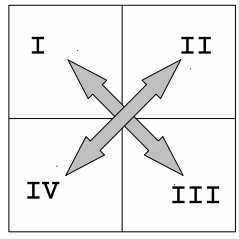
$$I_r = |I_r| \tag{7.8}$$

Maksymalne wartości transformaty są o kilka rzędów wielkości większe od jej mediany, zatem dla wygodniejszej percepcji transformaty warto ją zlogarytmować w celu kompresji zbyt dużej różnicy wartości:

$$I_r = \log(I_r + 1) \tag{7.9}$$

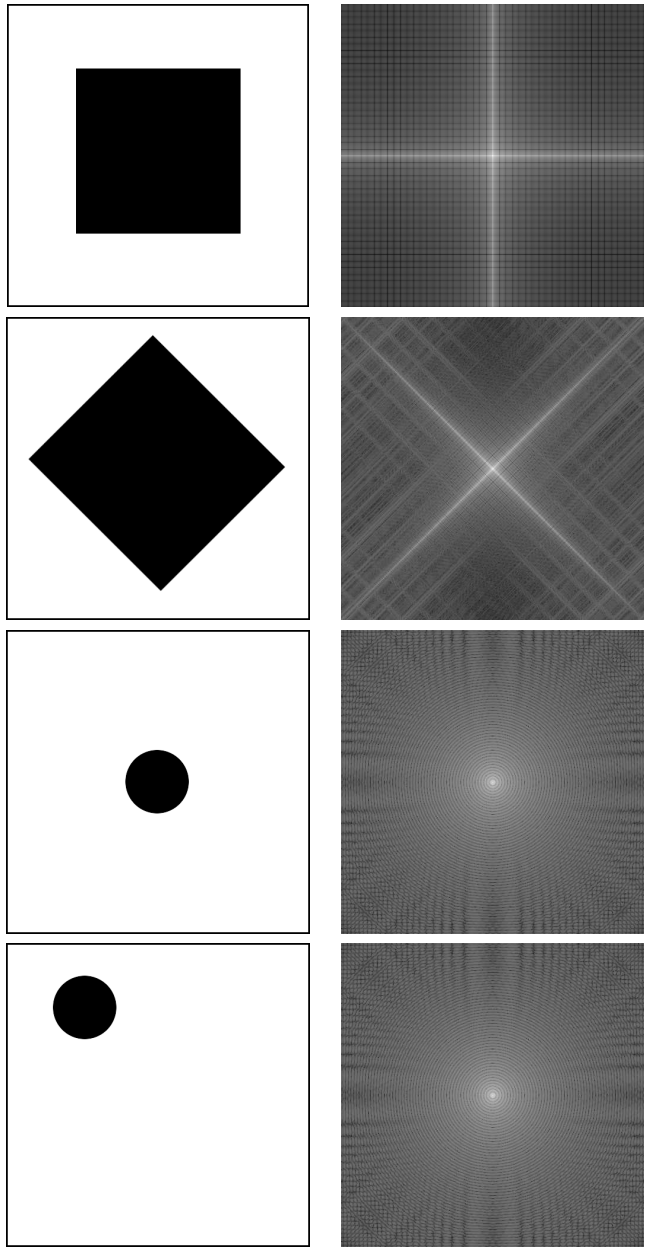
a następnie przeskalować I_r do zakresu [0..255] umożliwiającego wyświetlenie na ekranie w postaci obrazu.

W przypadku dwuwymiarowej transformaty zwykło się zamienić ćwiartki obrazu transformaty I z III i II z IV w identyczny sposób jak jest to zilustrowane na Rysunku 7.3. Dzięki takiemu zabiegowi najniższe częstotliwości znajdują się w środku transformaty i rosną wraz z oddalaniem się od centrum obrazu.



Rysunek 7.3. Zamiana ćwiartek w obrazie transformaty.

Rysunek 7.4 przedstawia przykładowe obrazy oraz ich transformaty.



Rysunek 7.4. Obraz (po lewej) i obraz modułu jego transformaty (po prawej). W obrazie transformaty zostały zamienione ćwiartki zgodnie z punktem 5.

7.5. Filtracja splotowa

Dla przeprowadzenia filtracji wykorzystujemy fakt, że operację splotu w dziedzinie obrazu można wykonać równoważnie w dziedzinie transformaty

poprzez przemnożenie transformat. To znaczy, dla danej maski splotu $h(x)$ i jej transformaty Fouriera $H(\omega)$ zachodzi poniższa relacja:

$$\mathcal{F}(f(x) \otimes h(x)) = \Phi(\omega)H(\omega) \tag{7.10}$$

gdzie operator $\otimes$ jest operatorem splotu w dziedzinie obrazu, a $\Phi(\omega)$ jest transformatą sygnału $f(x)$. Daje to możliwość zastąpienia operacji splotu z dziedziny obrazu mnożeniem w dziedzinie częstotliwości. Co więcej przejście do i z powrotem z dziedziny częstotliwości wymaga tylko dwóch transformat – wprost i odwrotnej i może być obliczone jako:

$$f(x) \otimes h(x) = \mathcal{F}^{-1}\left(\mathcal{F}\left(f(x)\right) \bullet \mathcal{F}\left(h(x)\right)\right) \tag{7.11}$$

gdzie operator $\bullet$ jest operatorem mnożenia tablicowego zdefiniowanego jako:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \bullet \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix} \tag{7.12}$$

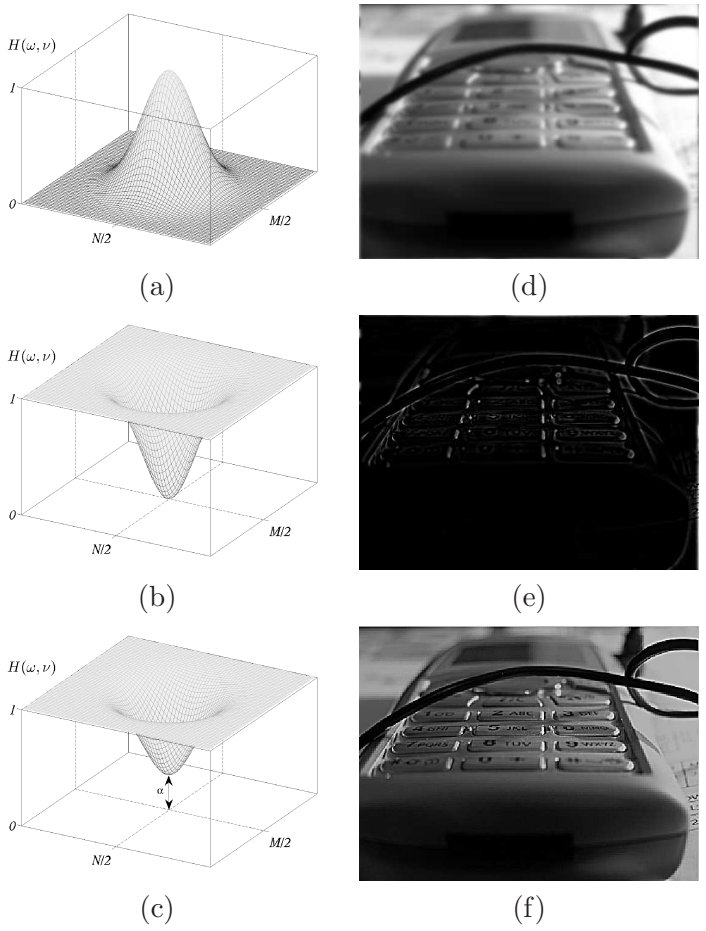
Warte zauważenia jest, że niskie częstotliwości w transformacie związane są z obszarami obrazu, w których następuje niewielka zmiana intensywności, wysokie częstotliwości natomiast z obszarami, w których są ostre przejścia w intensywności, takie jak krawędzie lub szum. W związku z tym, należy się spodziewać, że filtr $H(\omega, \nu)$, który tłumi wysokie częstotliwości, jednocześnie przepuszczając wysokie (filtr dolnoprzepustowy) powinien rozmywać obraz, podczas gdy filtr z odwrotną własnością (filtr górnoprzepustowy) powinien wzmacniać ostre detale, za cenę zredukowania kontrastu obrazu. Rysunek 7.5 ilustruje ten efekt. Dodanie niewielkiej wartości do filtru nie wpływa znacząco na poziom ostrości ale zapobiega wyeliminowaniu składowej średniej z obrazu i dzięki temu zachowuje ogólną tonację (Rysunek 7.5 (f)).

Niech I będzie obrazem wejściowym o rozmiarze (W, H), który zostanie poddany filtracji splotowej filtrem zdefiniowanym przez maskę M. Dla obrazów trójkanałowych (RGB) każdy kanał przeliczany jest osobno i niezależnie od pozostałych. Sam algorytm filtracji może być opisany następująco:

1. Wyznaczenie transformaty Fouriera dla obrazu I:

   $$F_I = fft(I) \tag{7.13}$$

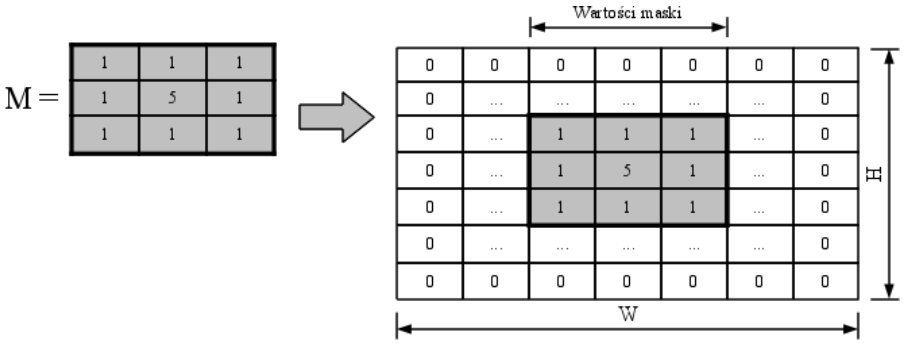
   Pełna Transformata Fouriera wymaga liczb zespolonych – obrazy wejściowe należy w takim razie potraktować jak liczby zespolone z częścią urojoną równą zero.
2. Przygotowanie maski filtru.
   Maskę należy rozszerzyć do rozmiaru obrazu (W, H). Istotne dane z maski muszą być umieszczone w centrum obrazu maski, nieistotne wartości



Rysunek 7.5. Filtracja w dziedzinie częstotliwości: (a) filtr $H(\omega, \nu)$ zmniejszający wartości wysokich częstotliwości oraz (b) rozmycie obrazu jako wynik splotu z tym filtrem, (c) filtr $H(\omega, \nu)$ pozostawia tylko wysokie częstotliwości oraz (d) obraz zawierający jedynie wysokie częstotliwości, (e) filtr $H(\omega, \nu)$ pozostawia wysokie częstotliwości przy niewielkim udziele składowej średniej $H = 0.7H + 0.3$ oraz (f) obraz przefiltrowany takim filtrem wyostrzającym.

w masce należy uzupełnić zerami. Przykładowa ilustracja przygotowania obrazu maski jest przedstawiona na Rysunku 7.6.

W tak przygotowanym obrazie maski należy zamienić ćwiartki I z III i II z IV zgodnie z ilustracją na Rysunku 7.3. Dla zachowania skali wartości filtrowanego obrazu można unormować wartości maski, tak żeby suma



Rysunek 7.6. Przykład przygotowania obrazu maski dla filtru.

wszystkich wartości maski była równa 1:

$$M(i) = \frac{M(i)}{\sum_k M(k)} \tag{7.14}$$

gdzie sumowanie po k oznacza sumowanie po wszystkich elementach maski.

3. Wyznaczenie transformaty Fouriera dla obrazu maski M:

$$F_M = fft(M) \tag{7.15}$$

4. Mnożenie obu wynikowych transformat:

$$F_{IM} = F_I \bullet F_M \tag{7.16}$$

gdzie operator mnożenia $\bullet$ jest zdefiniowany jako operator mnożenia składowych o tych samych indeksach w tablicy:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \bullet \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a \cdot e & b \cdot f \\ c \cdot g & d \cdot h \end{bmatrix} \tag{7.17}$$

Wartości transformaty są liczbami zespolonymi, zatem jeżeli:

$$z_1 = a + ib \quad \text{oraz} \quad z_2 = c + id \tag{7.18}$$

to wynik mnożenia liczby z_1 i z_2 jest równy:

$$z_1 \cdot z_2 = (a + ib) \cdot (c + id) = (ac - bd) + i(bc + ad) \tag{7.19}$$

5. Przeliczenie uzyskanej tablicy liczb zespolonych F_{IM} z dziedziny fourierowskiej za pomocą odwrotnej transformaty Fouriera na dziedzinę obrazu

$$I' = ifft(F_{IM}) \tag{7.20}$$

6. Z powstałej tablicy liczb zespolonych I' obraz jest zakodowany tylko w części rzeczywistej

$$I = real(I') \tag{7.21}$$

7.6. Kompresja stratna

Kompresja, słownikowo polega na takim przekształceniu zbioru liczb, żeby tę samą informację wyrazić za pomocą mniejszej ilości bitów. Kompresja stratna dodatkowo dopuszcza pewną utratę informacji z oryginalnego zbioru w zamian za jeszcze większe zmniejszenie objętości zbioru. Ten typ kompresji da się zastosować wszędzie tam gdzie nie ma potrzeby odwzorowania informacji 1:1, na przykład przy kodowaniu dźwięku lub obrazu gdzie odbiorca godzi się na pewną stratę jakości sygnału w zamian za zmniejszenie objętości strumienia danych [37]. Sam termin kompresji stratnej odnosi się do metody w której dane są początkowo przygotowywane poprzez usunięcie z nich nadmiarowych/akceptowalnych informacji a następnie kompresowane algorytmem kompresji bezstratnej.

W przypadku kompresji stratnej obrazu wykorzystuje się dwa zjawiska działania ludzkich zmysłów do redukcji nadmiarowej informacji. Pierwszym jest akceptacja utraty pewnych szczegółów obrazu, tj. elementów o wysokiej częstotliwości, często i tak nie dostrzegalnych przy normalnej percepcji. Rolę tę spełnia transformata kosinusowa i jej kwantyzacja. Drugim czynnikiem jest znacznie mniejsza czułość ludzkiego wzroku na zmianę barwy niż na zmianę kontrastu obrazu. Zatem informację o barwie można z powodzeniem zredukować nie tracąc przy tym na postrzeganiu danego obrazu. Ilustruje to Rysunek 7.7.

(a) (b)

Rysunek 7.7. Redukcja ilości informacji zawartych w kanałach chromatycznych modelu CIE L*a*b*. (a) – obraz oryginalny, (b) – obraz w którym zredukowano informację w kanałach A i B modelu CIE L*a*b* do 0,02% pierwotnej ilości.

Obraz oryginalny (Rysunek 7.7(a)) został przekonwertowany na model CIE L*a*b*. Następnie, rozdzielczość kanałów A i B została zmniejszona z oryginalnej 1024×768 do rozdzielczości 16×12, co daje 2 promile pierwotnej wielkości. W praktyce jest równoważne z niemalże całkowitym usunięciem

informacji z obu kanałów chromatycznych. Tak zmodyfikowany obraz został z powrotem przekonwertowany na model RGB, przy czym kanały A i B były interpolowane do pierwotnej wielkości metodą bikubiczną. Ilość potrzebnej informacji do zakodowania obrazu po takiej modyfikacji została zmniejszona do nieco ponad 1/3 jego pierwotnej wielkości, sam obraz natomiast stracił na jakości, chociaż nie na tyle na ile sugerowałaby to sama redukcja ilości informacji.

7.6.1. Kompresja

Niech I będzie obrazem, który ma zostać skompresowany o rozmiarze (W, H). Dla obrazów trójkanałowych (RGB) kompresję stratną można wykonać w następujących krokach:

1. Transformacja obrazu RGB na obraz w modelu CIE L*a*b*.

$$I_{Lab}(u, v) = \text{rgb2lab}(I(u, v)) \tag{7.22}$$

   Składowe L, a i b najlepiej przeskalować/przesunąć do zakresu jednego bajta bez znaku. Ewentualnie można użyć innego kodowania barw, które konwertuje model RGB na model trójkanałowy z jednym kanałem luminacji i dwoma kanałami chrominacji (np. CIE Luv, YCrCb).
2. Podział na bloki.
   Podział obrazu na logiczne bloki 8x8 punktów (lub inne potęgi dwójki). Blok nie musi być kwadratem. Kanał luminacji powinien być odwzorowywany 1 do 1, kanały chromatyczne mogą być przeskalowane w dół, czyli np. bloki 16x16 można zmniejszyć do rozmiaru 8x8 lub nawet bardziej.
3. Transformata kosinusowa.
   Dla każdego bloku dla każdego kanału obliczyć jego transformatę kosinusową:

$$T(u, v) = \text{dct}(I_{Lab}^{(8\times 8)}(u, v)) \tag{7.23}$$

4. Kwantyzacja.
   Dla każdego bloku każdego kanału należy wyznaczyć iloraz:

$$C(u, v) = \text{integer}\left(\frac{T(u, v)}{Q(u, v)}\right) \tag{7.24}$$

   gdzie $Q(u, v)$ jest macierzą kwantyzacji. Iloraz ten jest operacją dzielenia tablicowego a nie macierzowego, czyli dzielone są elementy o tych samych indeksach. Powstała macierz $C(u, v)$ ma wartości liczbowe całkowite. To w tym miejscu następuje największa strata jakości sygnału, a co za tym idzie możliwość największego zysku z kompresji, gdyż większość współczynników transformaty o wartościach ułamkowych, zostaje wyzerowana. Za wartości macierzy kwantyzacji można przyjąć standardowe

wartości kwantyzacji stosowane w standardzie JPEG [6] (wyrażenie 7.25 i 7.26) lub dowolne inne, według własnego uznania.
Dla kanału luminacji:

$$Q_L(u,v) = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{7.25}$$

Dla kanałów chrominacji:

$$Q_C(u,v) = \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix} \tag{7.26}$$

Stopniowanie jakości kompresji.
Niech współczynnik $QL \in (1, 100)$ odpowiada za poziom jakości kompresji (quality level). Wartość $QL = 50$ daje macierz kwantyzacji podaną w wyrażeniach 7.25 i 7.26.
Jeżeli $QL > 50$, wtedy macierz kwantyzacji $Q(u,v)$ przyjmuje wartości:

$$Q(u,v) = \frac{Q(u,v) * (100 - QL)}{50} \tag{7.27}$$

Jakość obrazu będzie wyższa, kosztem osłabienia kompresji.
Jeżeli $QL < 50$, wtedy macierz kwantyzacji $Q(u,v)$ przyjmuje wartości:

$$Q(u,v) = \frac{Q(u,v) * 50}{QL} \tag{7.28}$$

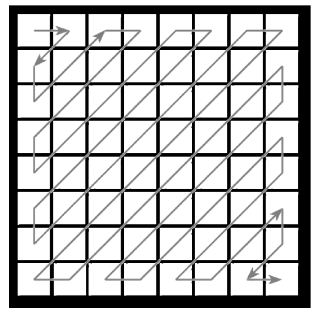
Jakość obrazu będzie niższa ale zwiększa się siła kompresji.

5. Kodowanie Zig-zag.
Przestawienie elementów macierzy $C(u,v)$ w kolejności opisanej przez macierz $Z(u,v)$ do liniowej tablicy $D(x)$ (zobacz Rysunek 7.8):

$$D(x) = \text{zigzag}(C(u,v), Z(u,v)) \tag{7.29}$$

gdzie:

$$Z(u,v) = \begin{bmatrix} 0 & 1 & 5 & 6 & 14 & 15 & 27 & 28 \\ 2 & 4 & 7 & 13 & 16 & 26 & 29 & 42 \\ 3 & 8 & 12 & 17 & 25 & 30 & 41 & 43 \\ 9 & 11 & 18 & 24 & 31 & 40 & 44 & 53 \\ 10 & 19 & 23 & 32 & 39 & 45 & 52 & 54 \\ 20 & 22 & 33 & 38 & 46 & 51 & 55 & 60 \\ 21 & 34 & 37 & 47 & 50 & 56 & 59 & 61 \\ 35 & 36 & 48 & 49 & 57 & 58 & 62 & 63 \end{bmatrix} \tag{7.30}$$



Rysunek 7.8. Schemat adresowania zig-zag.

Wartości macierzy $Z(u,v)$ opisują kolejność elementów z macierzy $C(u,v)$, w której mają być zapisane do tablicy $D(x)$. Takie przestawienie ma na celu przesunięcie wyzerowanych elementów na koniec bloku danych zwiększając moc algorytmów bezstratnej kompresji z punktu następnego.

6. Kompresja bezstratna.
   Kompresja bezstratna każdej tablicy $D(x)$ dowolnym algorytmem kompresji bezstratnej [40].
   Środowiska programistyczne dyskutowane w rozdziale 1 zawierają implementacje algorytmu ZIP:
   - Qt – wbudowana globalna funkcja kompresji zip

```
QByteArray qCompress(const QByteArray & data,
                     int compressionLevel = -1);
```

   oraz dekompresji zip:

```
QByteArray qUncompress(const QByteArray& data)
```

- Java – klasa `java.util.zip.ZipOutputStream` do kompresji strumienia danych oraz `java.util.zip.ZipInputStream` do dekompresji strumienia skompresowanych danych.
- C# – klasa `System.IO.Compression.GZipStream` do kompresji i dekompresji strumienia typu `System.IO.Stream`.

7.6.2. Dekompresja

Dekompresja obrazu, czyli odkodowanie wejściowego obrazu, następuje w odwrotnej kolejności do kodowania i może być opisane następującymi krokami:

1. Dekompresja bezstratna identycznym algorytmem jak w przypadku kompresji.
2. Odkodowanie zig-zag jednowymiarowych tablic $D(x)$ do dwuwymiarowych bloków skwantowanych wartości transformaty $C(u,v)$ na podstawie macierzy $Z(u,v)$.
3. Dekwantyzacja bloków. Przemnożenie tablicowe każdego bloku przez macierz kwantyzacji identyczną z użytą podczas kompresji:

$$T(u,v) = C(u,v) \bullet Q(u,v)$$

4. Odwrotna transformata kosinusowa poszczególnych bloków dająca w wyniku obraz zakodowany w modelu CIE L*a*b:

$$I_{Lab}^{(8\times 8)}(u,v) = \mathrm{idct}(T(u,v))$$

5. Złożenie całego obrazu z bloków.

$$I_{Lab}(u,v) = \cup \left\{ I_{Lab}^{(8\times 8)}(u,v) \right\}$$

6. Konwersja obrazu z modelu CIE L*a*b* na RGB.

$$I(u,v) = \mathrm{lab2rgb}(I_{Lab}(u,v))$$

7.6.3. Format pliku

Przy tak zdefiniowanym sposobie kompresji/dekompresji stratnej obrazu cyfrowego należało by zdefiniować własny format pliku przechowującego skompresowaną postać obrazu oraz operacje zapisujące i odczytujące (parsujące) taki plik. Środowiska programistyczne dyskutowane w rozdziale 1 zawierają pomocnicze klasy oraz metody strumieniowe ułatwiające operacje wejścia/wyjścia. Dla samego języka C++ metody strumieniowego zapisu/odczytu pliku zawiera klasa `std::fstream`, dla

bibliotek Qt będzie to klasa `QDataStream`. W środowisku Javy operacje strumieniowe na plikach można wykonać za pomocą klas dziedziczących z `java.io.InputStream/OutputStream`. Odpowiednio dla C# – klasa `System.IO.FileStream`.

Największą wadą zaproponowanej metody jest wprowadzanie do obrazu swoistego efektu kafelkowości na granicy bloków spowodowanej transformatą kosinusową. Nieznaczną poprawę może dać zwiększenie rozmiaru bloków kwantyzacji, większą zastosowanie zmodyfikowanej transformaty kosinusowej (MDCT), która przy obliczaniu transformaty bierze pod uwagę również informację znajdującą sie poza brzegami transformaty lub zastosowanie dyskretnej transformaty falkowej (DWT).

7.7. Watermarking

Osadzanie znaku wodnego (ang. *watermarking*) jest typowym sposobem zabezpieczania obrazu cyfrowego przed nieupoważnionym kopiowaniem. Najprostszym sposobem jest dodanie do obrazu widocznego znaku wodnego poprzez mieszanie obrazu oryginalnego i obrazu znaku wodnego z odpowiednią wartością przezroczystości.

$$f_w = (1 - \alpha)f + \alpha w \tag{7.31}$$

gdzie f_w jest oznakowanym obrazem, f obrazem oryginalnym, w obrazem znaku wodnego a α współczynnikiem kontrolującym przezroczystość a co za tym idzie widoczność znaku wodnego (zobacz Rysunek 7.9). Sposób jest bardzo prosty do wykonania ale niestety niesie ze sobą sporo wad. Przede wszystkim taki znak wodny jest bezpośrednio widoczny na obrazie i co za tym idzie łatwy do usunięcia. Po drugie nie jest zbyt odporny na transformacje obrazu, zarówno geometryczne jak i intensywności.

Nieco bardziej zaawansowanym sposobem znaczenia obrazów cyfrowych jest niewidoczny znak wodny (ang. *invisible watermark*). W tym przypadku chodzi o takie osadzenie znaku wodnego, który byłby niedostrzegalny dla percepcji zmysłowej lub systemów wizyjnych. Prostym przykładem tego typu osadzania jest wykorzystanie najmniej znaczących bitów dla 8-bitowych obrazów. Rozważmy równanie:

$$f_w = 4\left(\frac{f}{4}\right) + \frac{w}{64} \tag{7.32}$$

Całość obliczeń jest wykonywana z całkowitą precyzją arytmetyczną. Dzielenie i mnożenie przez 4 ustawia dwa najmniej znaczące bity wartości f na 0, dzielenie wartości w przez 64 przesuwa dwa najbardziej znaczące bity

Rysunek 7.9. Przykładowy obraz z osadzonym widocznym znakiem wodnym.

znaku wodnego na pozycję dwóch najmniej znaczących bitów. Zsumowanie tych wartości ostatecznie osadza znak wodny w w obrazie f. Istotną różnicą w stosunku do widocznego znaku wodnego jest jego odporność na celowe lub przypadkowe usunięcie. Niestety również ten sposób znakowania obrazu cyfrowego jest nieodporny na modyfikacje obrazu. Jakakolwiek modyfikacja intensywności, kompresja stratna lub transformacja geometryczna najczęściej spowoduje zniszczenie osadzonego znaku. Poza tym do weryfikacji oznakowania obrazu niezbędny jest osadzany znak wodny i odpowiedni system dekodujący obraz.

Najbardziej pożądanych sposobem osadzania znaku wodnego byłby zatem taki, który generuje obraz nierozpoznawalnie zmieniony w stosunku do oryginału, bez widocznego znaku wodnego i dodatkowo odporny na niezamierzone lub celowe ataki takie jak stratna kompresja, liniowa lub nieliniowa filtracja, dodanie szumu czy transformacje geometryczne obrazu. Takie założenia może spełniać osadzanie znaku wodnego nie w domenie obrazu ale w domenie częstotliwości. Do tego celu doskonale nadaje się transformata kosinusowa z jej dużą odpornością na zniekształcenia. Rozważmy następujący sposób osadzania znaku wodnego zaproponowanego przez Coxa w [9]:

1. Obliczenie dwuwymiarowej DCT dla obrazu $I(u, v)$

$$I_C(u, v) = \text{dct2}(I(u, v)) \tag{7.33}$$

2. Wyznaczenie K największych współczynników $c_1, c_2, ..., c_K$ transformaty I_C co do wartości bezwzględnej.
3. Stworzenie znaku wodnego poprzez wygenerowanie K-elementowej sekwencji pseudolosowej $w_1, w_2, ..., w_K$ z rozkładu Gaussa ze średnią wartością $\mu = 0$ i średnią wariancją $\sigma^2 = 1$.

4. Osadzenie znaku wodnego wygenerowanego w punkcie 3. do K największych komponentów DCT wyznaczonych w punkcie 2. zgodnie z równaniem:
$$c_i' = c_i \cdot (1 + \alpha w_i) \quad 1 \leqslant i \leqslant K \tag{7.34}$$
gdzie $\alpha > 0$ jest stałą, która kontroluje wpływ i siłę oddziaływania znaku wodnego w_i na współczynniki c_i. Nowe współczynniki transformaty c_i' mają zastąpić stare współczynniki c_i w transformacie I_C.
5. Powrót do przestrzeni obrazu poprzez obliczenie odwrotnej DCT ze zmodyfikowanej w punkcie 4. transformaty.
$$I'(u,v) = \text{idct2}(I_C'(u,v)) \tag{7.35}$$

(a) (b) (c)

Rysunek 7.10. Osadzanie znaku wodnego metodą transformaty kosinusowej: (a) obraz oryginalny, (b) obraz z osadzonym znakiem wodnym, (c) różnica obrazów (a) i (b) przeskalowana w intensywności do zakresu [0..255].

Rysunek 7.10 pokazuje różnicę pomiędzy obrazem oryginalnym i obrazem z osadzonym znakiem wodnym. Wizualnie oba obrazy są w zasadzie nierozróżnialne. Tak skonstruowany znak wodny jest jest dobrze chroniony z kilku powodów:

1) znak wodny jest generowany za pomocą liczb pseudolosowych bez wyraźnej struktury;
2) znak wodny jest osadzony w komponentach o różnych częstotliwościach, które mają wpływ na cały obraz w domenie przestrzennej zatem jego położenie nie jest oczywiste;
3) ataki na znak wodny często prowadzą do zniszczenia obrazu jako takiego, tzn. żeby wprowadzić zmiany w komponentach o istotnych częstotliwościach (w których jest znak wodny) muszą być duże zmiany w intensywnościach obrazu.

Procedura określania czy dany obraz jest zabezpieczony znakiem wodnym daje procentową pewność i wymaga znajomości wartości znaku wod-

nego $w_1, w_2, ..., w_K$ oraz komponentów DCT $c_1, c_2, ..., c_K$, w których został osadzony. Rozważmy następujący algorytm wykrywania znaku wodnego:

1. Obliczenie dwuwymiarowej DCT dla badanego obrazu.
2. Wybranie K współczynników DCT znajdujących się na pozycjach odpowiadających współczynnikom $c_1, c_2, ..., c_K$ w kroku 2. procedury osadzania znaku wodnego. Oznaczmy je przez $c''_1, c''_2, ..., c''_K$. Gdyby badany obraz był oznakowany bez żadnych modyfikacji wtedy każdy $c''_i = c'_i$ z punktu 4. kodowania znaku wodnego. W przypadku zmodyfikowanej kopii obrazu oznakowanego $c''_i \approx c'_i$ (c''_i będzie podobne do c'_i). W przeciwnym razie badany obraz będzie albo obrazem nieoznakowanym albo obrazem o zupełnie innym znaku wodnym.
3. Wyznaczenie znaku wodnego $w''_1, w''_2, ..., w''_K$:

$$w''_i = \frac{c''_i - c_i}{c_i} \tag{7.36}$$

4. Zbadanie podobieństwa oryginalnego znaku wodnego $w_1, w_2, ..., w_K$ wygenerowanego w punkcie 3. procedury osadzania z $w''_1, w''_2, ..., w''_K$ z poprzedniego punktu weryfikacji. Dobrą miarą podobieństwa jest korelacja wzajemna określona jako:

$$\gamma = \frac{\sum_{i=1}^{K}(w''_i - \overline{w''})(w_i - \overline{w})}{\sqrt{\sum_{i=1}^{K}(w''_i - \overline{w''})^2 \cdot \sum_{i=1}^{K}(w_i - \overline{w})^2}} \tag{7.37}$$

gdzie $\overline{w}$ i $\overline{w''}$ oznaczają średnie wartości K-elementowych znaków wodnych – oryginalnego i weryfikowanego odpowiednio.

Wartość korelacji wzajemnej obrazu oznakowanego samego ze sobą będzie miała wartość $\gamma = 0.99$. Niewątpliwie oznacza to całkowitą zgodność obrazów znaku wodnego. Wartość bliska zeru korelacji oznacza, że testowany obraz nie został oznaczony danym znakiem wodnym.

Rysunek 7.11 przedstawia kilka przykładów ataków na obraz oznakowany wraz z wartością korelacji γ w stosunku do obrazu oryginalnego. Dla osadzenia znaku wodnego przyjęto parametry o wartościach: $K = 1000$, $\alpha = 0.1$. Filtracja splotowa filtrem uśredniającym o niewielkich rozmiarach nie powoduje praktycznie żadnej zmiany znaku wodnego. Istotną zaletą tego typu osadzania znaku wodnego będzie również jego duża odporność na operację kompresji stratnej, będącej częstym przykładem niecelowego ataku na znak wodny. W rozważanym przykładzie obraz został mocno skompresowany ze współczynnikiem jakości ustawionym na 50%. Wizualnie obraz

został zdegradowany, łącznie z pojawieniem się charakterystycznych blokowych artefaktów. Mimo wszystko wartość korelacji znaku wodnego daje niemalże 100% pewność oznakowania obrazu. Nieco mocniej na wartość korelacji wpływa wyrównywanie histogramu obniżając jej wartość do nieco ponad 0.5. Taka wartość mimo wszystko daje dużą pewność oznakowania obrazu. Binaryzacja wartości korelacji, czyli progowanie jej na pewnej ustalonej wartości ułatwiłaby weryfikację oznakowania.

(a) (b) $\gamma = 0.98$

(c) $\gamma = 0.96$ (d) $\gamma = 0.56$

Rysunek 7.11. Przykłady ataków na znak wodny i wartość korelacji obrazu oznakowanego: (a) obraz oryginalny, (b) obraz przefiltrowany filtrem uśredniającym 3×3, wartość korelacji $\gamma = 0.98$, (c) obraz skompresowany algorytmem JPEG z jakością 50%, wartość korelacji $\gamma = 0.96$, (d) obraz z wyrównanym histogramem, wartość korelacji $\gamma = 0.56$.

Bibliografia

[1] Adobe Systems Inc., *Adobe RGB (1998) Color Image Encoding*, http://www.adobe.com/digitalimag/pdfs/AdobeRGB1998.pdf.

[2] Bednarczuk, J., *Urok przekształceń afinicznych*, WSiP, Warszawa, 1978.

[3] Blanchette, J., Summerfield, M., *C++ GUI Programming with Qt 4 (Second Edition)*, Prentice Hall Open Source Software Development Series, Prentice Hall, 2008.

[4] Brigham, E.O., *The Fast Fourier Transform and its Applications*, Prentice Hall, Upper Saddle River, New York, 1998.

[5] Burger, W., *Digital Image Processing: An Algorithmic Introduction using Java*, Springer; 1 edition, 2007.

[6] CCITT, *Information Technology – Digital Compression and Coding of Continuous-tone Still Images – Requirements and Guidelines*, Recomendation T.81, The International Telegraph and Telephone Consultative Committee, ITU, 1993.

[7] Champeney, D.C., *A Handbook of Fourier Theorems*, Cambridge University Press, New York, 1983.

[8] CIE, *Commission internationale de l'Eclairage proceedings*, Cambridge University Press, Cambridge, 1932.

[9] Cox, I., Kilian, J., Leighton, F., Shamoon, T., *Secure Spread Spectrum Watermarking for Multimedia*, IEEE Trans. Image Proc., vol. 6, no. 12, 1997, s.1673-1687.

[10] Cox, I., Miller, M., Bloom, J., *Digital Watermarking*, Morgan Kaufmann (Elsevier), New York, 2001.

[11] Curcio, C.A., Sloan, K.R., Kalina, R.E., Hendrickson, A.E.. *Human photoreceptor topography*, The Journal of comparative neurology. Wiley-Liss, Feb 22;292(4), 1990, s. 497-523.

[12] Eckel, B., *Thinking in Java. Edycja polska. Wydanie IV*, Helion, 2006.

[13] Fairchild, M.D., Berns, R.S., *Image color-appearance specification through extension of CIELAB*, Color Research & Application, Volume 18, Issue 3, 1993, 178-190.

[14] Fairman, H.S., Brill, M.H., Hemmendinger H., *How the CIE 1931 Color-Matching Functions Were Derived from the Wright–Guild Data*, Color Research and Application 22 (1), John Wiley & Sons, Inc., 1997, s. 11–23.

[15] Frigo, M., Johnson, S.G.. *Fastest Fourier Transform in the West*, MIT, http://www.fftw.org.

[16] Gonzalez, R.C., Woods, E., *Digital Image Processing*, Third Edition, Pearson Prentice Hall, 2008.

[17] Gonzalez, R.C., Woods, E., Eddins, S.L., *Digital Image Processing Using MATLAB*, Prentice Hall, 2003.
[18] Harris, A.C., Weatherall, I. L., *Objective evaluation of colour variation in the sand-burrowing beetle Chaerodes trachyscelides White (Coleoptera: Tenebrionidae) by instrumental determination of CIE LAB values*, Journal of the Royal Society of New Zealand (The Royal Society of New Zealand), 1990.
[19] Heijden, F., *Image Based Managment System*, Wiley & Sons, 1995.
[20] International Color Consortium, *Specification ICC.1:2004-10*, http://www.color.org.
[21] Jähne, B., Haußecker H., *Computer Vision and Applications. A Guide for Students and Practitioners*. Academic Press, San Diego, 2000.
[22] Jähne, B., *Digital Image Processing*, 5th edition, Springer-Verlag, Berlin Heidelberg, 2002.
[23] Kuczyński, K., Denkowski, M., Mikołajczak, P., Stęgierski, R., Dmitruk, K., Pańczyk, M., *Wstęp do programowania w Qt*, Instytut Informatyki UMCS, Lublin, 2012.
[24] Lindbloom, B., *3D representations of the L*a*b* gamut*, http://www.brucelindbloom.com/index.html?WorkingSpaceInfo.html.
[25] Lohmeyer, M.S., *Digital Image Warping: Theory and Real Time Hardware Implementation Issues*, Massachusetts Institute of Technology, 1996.
[26] Matheron, G., *Random Sets and Integral Geometry*, Wiley, New York, 1975.
[27] Minkowski, H., *Volumen und Oberfläche*, Mathematische Annalen, 57, 1903, s. 447–459.
[28] Moszyńska, M., Święcicka, J., *Geometria z algebrą liniową*, Państwowe Wydawnictwo Naukowe, Warszawa 1987.
[29] Nienawski, M., *Morfologia matematyczna w przetwarzaniu obrazów*, Akademicka Oficyna Wydawnicza PLJ, Warszawa, 1998.
[30] Paris, S., Kornprobst, P., Tumblin, J., Durand, F., *Bilateral Filtering Theory and Application*, Now Publishers Inc. Hanover, MA, USA, 2009.
[31] Pavlidis, T., *Grafika i przetwarzanie obrazów*, Wydawnictwa Naukowo-Techniczne, Warszawa, 1987.
[32] Pennebaker, W.B., Mitchell, J.L., *Jpeg: Still Image Data Compression Standard*, Kluwer Academic Publishers, 1993.
[33] Poynton, C.A., *A Technical Introduction to Digital Video*, J. Wiley & Sons, New York, 1996.
[34] Pratt, W.K., *Digital Image Processing*, John Wiley and Sons, Inc., New York, 2001.
[35] Press, W.H., Teukolsky. S.A., Vetterling, W.T., Flannery, B.P., *Numerical Recipes, The Art of Scientific Computing*, Third Edition, Cambridge University Press, New York, 2007.
[36] Prewitt, J.M.S., *Object Enhancement and Extraction*, in *Picture Processing and Psychopictorics*, Lipkin, B.S., Academic Press, New York, 1970.
[37] Rabbani, M., Jones, P.w., *Digital Image Compression Techniques (SPIE Tutorial Text Vol. TT07) (Tutorial Texts in Optical Engineering)*, SPIE Publications, 1991.
[38] Roberts, L.G., *Machine Perception of Three-Dimensional Solids* in *Optical and*

Electro-Optical Information Processing, Tippet, J.T., MIT Press, Cambridge, Mass, 1965.

[39] Robinson, K., Whelan, P.F., *Efficient morphological reconstruction: a downhill filter*, Pattern Recognition Letters, Volume 25, Issue 15, 2004, s. 1759 - 1767.

[40] Salomon, D., Motta, G., *Handbook of Data Compression*, Springer, 5th ed. edition, 2009.

[41] Serra, J., *Image analysis and mathematical morphology*, Academic Press, London, 1982.

[42] Serra, J., Soille, P., *Mathematical Morphology and its Applications to Image Processing*, vol. 2 of Computational Imaging and Vision. Kluwer, Dordrecht, 1994.

[43] Schanda, J., *Colorimetry Understanding The CIE System*, J. Wiley & Sons, 2007.

[44] Sharma, G., *Digital color imaging handbook*, CRC Press, 2003.

[45] Sobel, I.E., *Camera Models and Machine Perception*, Ph.D. dissertation, Stanford University, Palo Alto, California, 1970.

[46] Soille, P., *Morphological Image Analysis. Principles and Applications*, Springer, Berlin, 1999.

[47] Steinberg, S.R., *Biomedical Image Processing*, IEEE Computer, January 1983, s. 22–34.

[48] Tadeusiewicz, R., Korohoda, P., *Komputerowa analiza i przetwarzanie obrazów*, FPT, Kraków, 1997.

[49] Tadeusiewicz, R., *Systemy wizyjne robotów przemysłowych*, Wydawnictwa Naukowo-Techniczne, WArszawa, 1992.

[50] Tadeusiewicz, R., Flasinski, M., *Rozpoznawanie obrazów*, PWN 1991.

[51] Tomasi, C., Manduchi, R., *Bilateral Filtering for gray and color images*, Sixth International Conference on Computer Vision, New Delhi, India, 1998, s. 839-846.

[52] Watkins, C.D., Sadun, A., Marenka, S., *Nowoczesne metody przetwarzania obrazu*, Wydawnictwa Naukowo-Techniczne, Warszawa, 1995.

[53] Wolberg, G., *Digital Image Warping*, Wiley-IEEE Computer Society Press, 1990.

[54] Zabrodzki, J. i inni. *Grafika komputerowa metody i narzędzia*, Wydawnictwa Naukowo Techniczne, Warszawa, 1994.

Skorowidz