PRACA DOKTORSKA

System wspomagania orientacji przestrzennej osób niewidomych wykorzystujący konwersję obrazów 2D do postaci dźwięku przestrzennego

Autor: *mgr inż. Robert Świta*
Promotor: *prof. dr hab. inż. Zbigniew Suszyński*

Wydział Elektroniki i Informatyki
Politechnika Koszalińska 2015

Dziękuję swojemu promotorowi,
prof. Z.Suszyńskiemu za
nieocenioną pomoc w relizacji
niniejszej pracy

Spis treści

1. Wstęp

1.1 Teza i cele pracy

Przedmiotem dysertacji jest metoda przetwarzania danych wizualnych 2D w celu transformacji wybranych treści obrazu do postaci dźwięku przestrzennego. Przetwarzanie to powinno wspomagać orientację przestrzenną osób niewidomych. Przez orientację przestrzenną autor rozumie możliwość określenia z ograniczoną dokładnością kształtu, rozmiaru, położenia, oraz odległości obserwatora do analizowanych obiektów sceny.

W niniejszej pracy podjęto próbę udowodnienia następującej tezy: „Przetwarzanie informacji wizualnej 2D wykorzystującej akwizycję obrazów RGB, ich segmentację oraz transformację kształtu obiektów sceny do postaci dźwięku przestrzennego pozwala analizować kształt, położenie oraz szacować odległość pomiędzy obserwatorem a obiektem".

Cele pracy:

- Udoskonalenie algorytmów segmentacji obrazów RGB w czasie rzeczywistym.
    - Wyznaczenie funkcji porównującej piksele kolorowe przy nierównomiernym oświetleniu obiektów światłem białym
    - Opracowanie algorytmu automatycznej regulacji balansu bieli
    - Opracowanie efektywnych metod segmentacji obrazu w oparciu o segmentację obszarową.
        - Połączenie zalet metod segmentacji wododziałowej i rozrostowej
        - Implementacja segmentacji klasterowej k-means oraz split & merge
- Opracowanie metody transformacji treści obrazu 2D do postaci dźwięku przestrzennego z zastosowaniem funkcji HRTF (Head Related Transform Function) oraz standardu MIDI i wykorzystania jej do analizy kształtów, położenia oraz szacowania odległości pomiędzy obserwatorem i obiektem.
- Opracowanie metody rozpoznawania wybranych obiektów sceny na podstawie ich kształtu i tekstury z użyciem SSN.
- Napisanie programu do weryfikacji przyjętych tez i opracowanych algorytmów.

Oryginalne osiągnięcia autora:

- Opracowanie metody transformacji informacji o kształcie, rozmiarach i położeniu obiektu do postaci dźwięku przestrzennego. Zaproponowano, by informacja o obrazie zawarta była zarówno w wysokości dźwięku jak i położeniu przestrzennym jego źródła.

- Zdefiniowanie przestrzeni kolorów HCI, w której możliwe jest zmniejszenie wpływu nierównomiernego oświetlenia światłem białym na segmentację obiektów.
- Określenie funkcji porównującej piksele na podstawie ich koloru w przestrzeni HCI. Jest to funkcja różnic parametrów HCI kolorów pikseli, w której wagami są funkcje dwuargumentowe – powierzchnie interpolacji biliniowej parametrów zależnych od stosunku nasyceń do progu rozpoznania koloru.
- Modyfikacja algorytmu interpolacji bikubicznej wykorzystującego płaty powierzchni sklejanych Beziera.
- Opracowanie nowego sposobu parametryzacji krzywych B-sklejanych.
- Opracowanie algorytmu automatycznej korekcji balansu bieli obrazu, łączącego zalety metod Gray Edge i Shades of Gray.
- Opracowanie algorytmu segmentacji wododziałowo-rozrostowej, która łączy zalety obu rodzajów segmentacji obszarowej.
- Modyfikacja metody jednostronnego przekształcenia Jacobiego do wyznaczania faktoryzacji SVD macierzy
- Opracowanie nowej metody inicjalizacji segmentacji klasterowej obrazów kmeans, uwzględniającej rozkład gęstości pikseli (High Density)
- Opracowanie modułu wspomagającego klasyfikację obiektów sceny z wykorzystaniem sieci neuronowych i bloków PCA, redukujących liczbę wejść sieci SSN i ułatwiających jej naukę.
- Propozycja metody szacowania odległości pomiędzy obserwatorem a obiektem, polegającej na analizie zmiany wysokości dźwięku towarzyszącej przemieszczeniu się obserwatora względem obiektu.
- Opracowanie metody dekompozycji (faktoryzacji) dowolnej transformacji afinicznej obiektów edytora sceny wirtualnej do postaci złożenia macierzy transformacji podstawowych w kolejności: skalowanie, pochylenie, obrót i przesunięcie (SHRT).

1.2 Układ pracy

Praca podzielona jest na osiem rozdziałów, ułożonych w sposób zgodny z kolejnością przetwarzania danych i zaawansowania użytych narzędzi. Rozdziały opatrzone są licznymi ilustracjami i istotnymi fragmentami kodu. Użyta składnia jest przeważnie zgodna z obiektowym językiem C# (niekiedy ze wskazaniem w metodach nazwy klasy, do której należą) lub językiem programowania w środowisku MATLAB - jeśli struktura obiektowa danych lub programu nie była szczególnie istotna. Poszczególne rozdziały zawierają następujące treści:

Rozdział pierwszy

Stanowi przegląd i podział istniejących systemów słyszenia obrazów. Zaprezentowana została koncepcja proponowanego rozwiązania i budowa systemu konwersji obraz-dźwięk.

Rozdział drugi

W rozdziale drugim przedstawione zostały podstawy teorii koloru i kolorymetrii, oraz przestrzeń kolorów użyta do porównywania pikseli podczas segmentacji obrazu. Zdefiniowana została miara odległości pomiędzy kolorami w tej przestrzeni, umożliwiająca zmniejszenie wpływu nierównomiernego oświetlenia obiektów światłem białym na segmentację obrazu.

Rozdział trzeci

Omawia sposoby interpolacji obrazów, przedstawia metodę tworzenia palet barw i implementację klasy *Pixelmap* umożliwiającej wygodne posługiwanie się obrazami i paletami pseudokolorów. Przedstawione zostały podstawowe metody automatycznego balansu bieli i zaproponowane połączenie ich zalet w nowym rozwiązaniu.

Rozdział czwarty

Na wstępie rozdziału czwartego zaprezentowane zostały podstawy matematyczne analizy czynników głównych PCA i faktoryzacji SVD macierzy. Metoda PCA umożliwia redukcję i filtrację danych poprzez wyznaczenie cech, które są od siebie liniowo niezależne i znajduje szerokie zastosowanie w systemach przetwarzania. Obok znanych i powszechnie używanych algorytmów, przedstawione zostały sugestie i modyfikacje autora tych algorytmów. Zasadniczą część rozdziału czwartego stanowi jednak opis wybranych metod segmentacji obszarowej oraz segmentacji hybrydowej, która powstała z połączenia popularnych metod segmentacji rozrostu obszaru oraz segmentacji wododziałowej. Rezultat działania tych metod został porównany z innymi rodzajami segmentacji z wykorzystaniem projektu Uniwersytetu Berkeley BSDS500.

Rozdział piąty

Przedstawia opracowaną realizację rozpoznawania obiektów z użyciem sieci neuronowych. Moduł wykorzystuje dwie sieci neuronowe poprzedzone blokami PCA.

Rozdział szósty

Przedstawia opracowaną metodę generowania dźwięku na podstawie obrazu segmentu. Kształt i położenie segmentu podlega transformacji na dźwięk przestrzenny przy użyciu szybkiej konwolucji dźwięku z funkcją HRTF dla ucha prawego i lewego. Omówiona została również możliwość szacowania odległości i wielkości obiektów

przez obserwatora na podstawie szybkości zmian częstotliwości dźwięku w zależności od odległości od obiektu.

Rozdział siódmy

Aplikacja wyposażona jest również w moduł przeznaczony do nauki systemu, którego opis znajduje się w rozdziale siódmym. Umożliwia on tworzenie dowolnie skomplikowanych kształtów dzięki funkcji grupowania i rozgrupowywania kształtów prostych. Kształty oraz ich grupy mogą podlegać transformacjom afinicznym 2D i podlegać konwersji na dźwięk przestrzenny.

Rozdział ósmy

Stanowi zbiór zebranych wniosków, wynikających z przeprowadzonych badań, implementacji i eksperymentów, zakończony podsumowaniem.

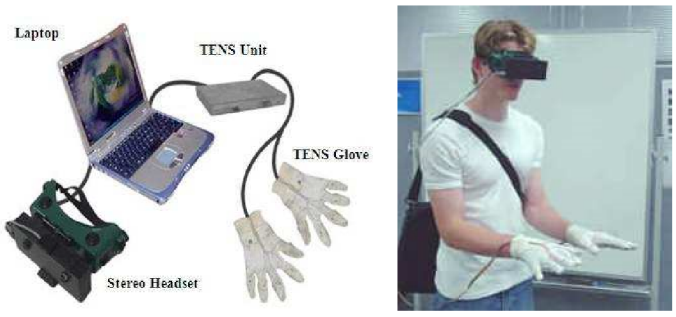
Dodatek A

Zawiera listę dostępnych instrumentów w standardzie General MIDI.

Dysertację zamyka wykaz literatury, do której znalazły się odnośniki w tekście.

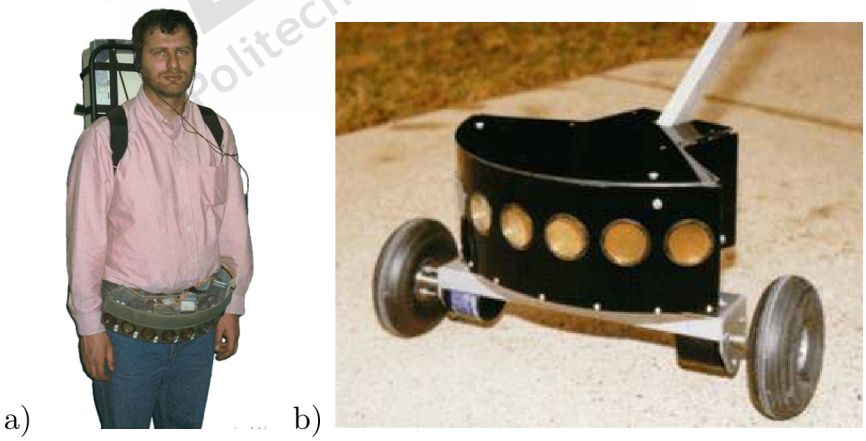
1.3 Aktualny stan wiedzy

Znakomita większość docierających do nas bodźców jest rejestrowana za pomocą oczu. Przetwarzanie tak dużej liczby informacji nie jest proste. Nic dziwnego, że jedna trzecia naszych mózgów jest związana z tą podstawową dla nas możliwością - widzeniem. Tym samym terminem „widzenia" określamy bowiem zazwyczaj nie tylko kwestię rejestracji obrazów, ale również rozpoznawania widocznych na nich obiektów. Utrata wzroku, jako najważniejszego ze zmysłów, wiąże się często z drastycznym ograniczeniem interakcji człowieka z otoczeniem i obniżeniem komfortu życia. Światowa Organizacja Zdrowia szacuje liczbę osób z ciężkim upośledzeniem wzroku na około 290 milionów (sierpień 2014), z czego 40 milionów to osoby całkowicie niewidzące. W podwyższeniu samodzielności i warunków życia osób dotkniętych tą niepełnosprawnością dużą rolę odgrywają urządzenia konwersji obrazów na sygnały postrzegane innymi receptorami. Naturalnym wyborem jest często słuch, którym odbieramy większość pozostałych informacji z otoczenia. Istnieją jednak systemy, które starają się wykorzystać możliwość przekazania dwuwymiarowej informacji o obrazie za pomocą zmysłu dotyku i elektrod umieszczonych w postaci matrycy. W obawie przed uciążliwością sygnałów dźwiękowych i maskowaniem informacji audio z otoczenia wykorzystywane są również często urządzenia wibrujące [1] [2] (Rys. 1).



Rys. 1. System ENVS konwersji obraz – dotyk. © [2004] IEEE

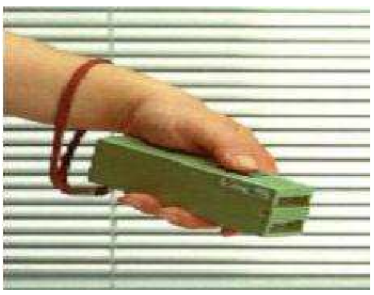
Istnieje wiele systemów wspomagających nawigację osób niewidomych, określanych wspólnym mianem ETA (Electronic Travel Aid). Pierwsze takie systemy, powstałe w latach 70-tych oparte były na aktywnym skanowaniu odległości od przeszkód. W zależności od użytego sensora można je podzielić na radary, ladary (używające lasera) i sonary emitujące ultradźwięki. Przykładem ladaru jest urządzenie LaserCan, które umożliwia detekcję obiektów na poziomie głowy, stromych spadków terenu oraz obiektów w odległości do 3.5 m przed użytkownikiem przy pomocy trzech par diód laserowych i fotodiód [3]. Bardzo popularne są czujniki wykorzystujące ultradźwięki ze względu na możliwość symulowania echa odbić dźwięku od przeszkody. Sonarami są m.in. urządzenia BatCane, NavBelt [4] i GuideCane [5] [6] - rodzaj laski na kółkach z sensorami ultradźwiękowymi, która omija przeszkody prowadząc bezpiecznie osoby niewidzące (Rys. 2). GuideCane może również korzystać z informacji GPS lub wykorzystywać wcześniej zdefiniowaną ścieżkę ruchu.



Rys. 2. Sonary ETA: Urządzenie NavBelt składa się z ośmiu ultradźwiękowych czujników odległości, komputera przetwarzającego informację na dźwięk i słuchawek (a). Głowica sonaru GuideCane (b). © [2003] IEEE

Czujnik Mowat [7] sygnalizuje odległość od przeszkody wibracjami o częstotliwościach odwrotnie proporcjonalnych do odległości. Niewidomy rejestruje zatem częstsze wibracje zbliżając się do przeszkody. Na podobnej zasadzie działa Nottingham Obstacle Detector (NOD) [8] i Sonic Torch [9] (Rys. 3), sygnalizując obiekty za

pomocą ośmiu różnych tonów dźwiękowych, oraz Polaron, który potrafi sygnalizować zarówno dźwiękiem jak i wibracjami.

a)  b) 

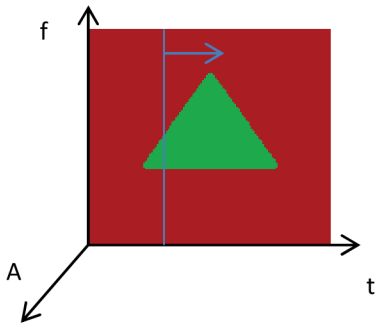
Rys. 3. Electronic Travel Aids: Mowat Sensor (a), Sonic Torch (b)

Zainspirowany sposobem echolokacji nietoperzy jest system skanowania sonarowego SonicGuide (autorstwa prof. Leslie Kay'a), z nadajnikiem ultradźwiękowym i dwoma receptorami przekształcającymi echo w stereofoniczny dźwięk umożliwiający detekcję najbliższej przeszkody [10]. Zestaw oryginalnie umieszczony w specjalnych okularach został zmodyfikowany w opaskę na czoło w systemie KASPA. Urządzenia te potrafiły już przekazać większą informację o otoczeniu niż kierunkowy sygnał wcześniejszych ETA. W zaawansowanych urządzeniach przetwarzających obraz możliwe jest nie tylko ostrzeganie przed przeszkodami, ale również konwersja wspomagająca modelowanie otoczenia i orientację przestrzenną użytkowników systemu.

a)  b) 

Rys. 4. Urządzenie SonicGuide (a) i jego odmiana – KASPA (b)

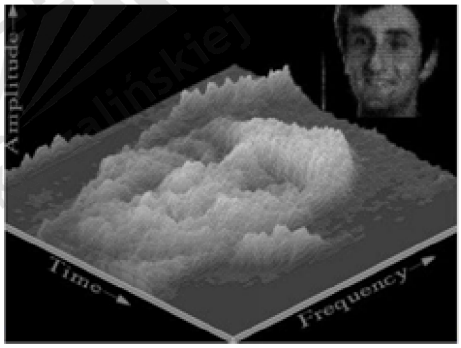
Współczesne systemy przetwarzania obrazów na dźwięk można podzielić na dwie zasadnicze grupy. Do pierwszej należą algorytmy przetwarzania obrazów rastrowych wykorzystujące analizę częstotliwościową obrazów. Każda kolumna obrazu traktowana jest wówczas jako widmo amplitudowe dźwięku i odgrywana na zasadzie skanowania przetwarzanego obrazu (Rys. 5).



Rys. 5. Ilustracja konwersji ze skanowaniem obrazu

Przykładem komercyjnym takiego rozwiązania jest holenderski skaner The vOICe opatentowany przez Petera Meijer'a [11]. Niskie częstotliwości generowanego dźwięku odpowiadają dolnym wierszom obrazu, a wyższe – górnym (Rys. 6). Rozwiązanie jest proste i jednoznacznie odzwierciedla pełną informację o obrazie. Ze względu jednak na złożoność informacji, system jest dość trudny do nauczenia, dodatkowo uniemożliwia on normalne słyszenie dźwięków docierających z otoczenia, zastępując całkowicie słuch, protezą zmysłu wzroku.

a) 

b) 

Rys. 6. System The vOICe (a) i obraz zakodowany widmem dźwięku (b)

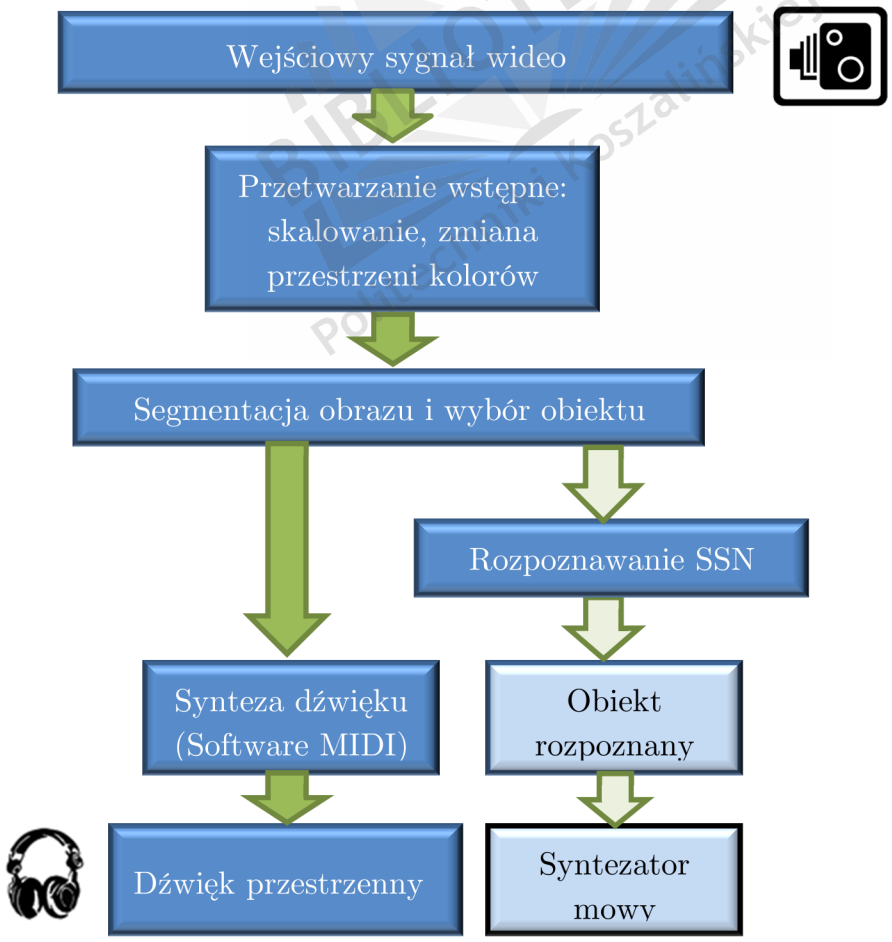
Drugą grupę stanowią systemy rozpoznające obiekty na obrazie w procesie segmentacji, poprzez grupowanie pikseli o wspólnych cechach. Na Uniwersytecie La Laguna w Hiszpanii (Teneryfa) skonstruowano urządzenie EAV (Espacio Acustico Virtual) składające się z dwóch kamer i odwzorowujące obrazy w scenę trójwymiarową, w której każdy rozpoznany obiekt jest źródłem dźwięku [12]. Podobny projekt pod nazwą Naviton zrealizowany został na Politechnice Łódzkiej [13], oraz na Universytecie Sabah w Malezji (urządzenie SVETA [14], Rys. 7a) i Uniwersytecie Genewskim (See ColOr [15], Rys. 7b). Projekty EAV i Naviton generują dźwięk przestrzenny z wykorzystaniem funkcji HRTF na podstawie środka geometrycznego obiektu i jego wielkości (powierzchni lub objętości). Tego rodzaju rozwiązania stosowane są w ujawnionych patentach US3800082 (A), US20130038600 (A1), US8797386 (B2), US20070211947 (A1) i US20050208457 (A1). Omawiane systemy nie analizują kształtu obiektów, a jedynie ich wielkość i położenie.

a)  b) 

Rys. 7. Prototyp systemu SVETA (a) i See ColOr (b)

1.4 Koncepcja metody transformacji obrazu

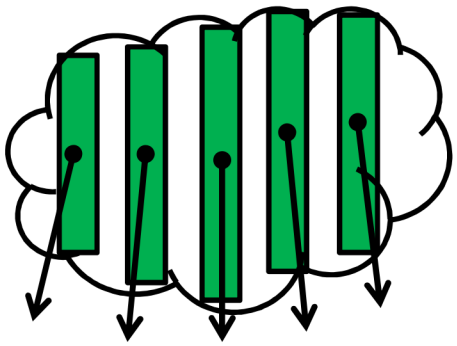
Założeniem pracy doktorskiej było opracowanie metody transformacji obrazu do postaci dźwięku przestrzennego, w celu wspomagania orientacji przestrzennej osób niewidomych. W metodzie (Rys. 8) poddaje się analizie sekwencję obrazów rejestrowanych przez kamerę.



Rys. 8. Schemat ogólny konwersji obraz - dźwięk

Z pojedynczych kadrów, po przetwarzaniu wstępnym oraz segmentacji, wydzielane są obiekty, które poddawane są dalszej analizie. Zawiera ona dwie niezależne ścieżki – transformację kształtu segmentu do postaci dźwięku przestrzennego oraz próbę rozpoznania obiektu na podstawie kontekstu, kształtu i tekstury. W pracy skoncentrowano się na realizacji ścieżki pierwszej - metodzie transformacji obrazu do postaci dźwięku. Dźwięk poddawany jest transformacji wykorzystującej funkcję HRTF, co umożliwia m.in. lokalizację przestrzenną źródła. Metoda korzysta z popularnego interfejsu MIDI do generowania dźwięku instrumentów muzycznych w celu zapewnienia harmonii i większego komfortu niż przy użyciu przypadkowych dźwięków syntetycznych, o dowolnej charakterystyce tonalnej. Druga ścieżka ma charakter wspomagający, a jej funkcja polega na rozpoznawaniu kształtu oraz tekstury analizowanego obiektu i ewentualne jego rozpoznanie w określonym kontekście (konkretne pomieszczenie). W przypadku rozpoznania obiektu przez SSN osoba niewidoma usłyszy komunikat z nazwą obiektu.

Segment poddawany konwersji na dźwięk jest reprezentowany w postaci kolumn (Rys. 9). Kolumny w kolejności od lewej do prawej są transformowane na tony instrumentu muzycznego. Generowany dźwięk zależy od liczby kolumn, wysokości i położenia ich środków oraz czasu odtwarzania. Liczba kolumn i czas odtwarzania dźwięku jest stała dla każdego segmentu i jest parametrem przetwarzania. Wysokość kolumn i położenie ich środków jest związane bezpośrednio z wielkością i kształtem transformowanego obiektu. Wysokość kolumny decyduje o wysokości tonu instrumentu, natomiast środek kolumny jest interpretowany jako położenie źródła dźwięku. W ten sposób na podstawie dźwięku możliwe jest określenie kształtu i położenia obiektu. Ze względu na konieczność rozróżniania położenia źródła dźwięku w pionie, do generacji dźwięku wykorzystana została funkcja transmitancji HRTF. Dopasowanie jej charakterystyk do cech osobniczych wymaga treningu. Wraz ze zwiększeniem szybkości rozpoznawania możliwe jest jednak skrócenie czasu odtwarzania dźwięku związanego z obiektem.



Rys. 9. Sposób konwersji segmentu na dźwięk przestrzenny

2. Teoria koloru

2.1 Przestrzenie kolorów

Kolor, czyli barwa, jest wrażeniem uzyskiwanym dzięki analizie widma spektralnego światła docierającego do oka. Najczęściej do opisu tego wrażenia wystarczają trzy niezależne parametry. Urządzenia rejestrujące barwę światła określają widmo poprzez zapamiętanie poziomów składowych R, G i B. Modele koloru związane z fizycznymi parametrami promieniowania, określającego widmo energetyczne światła, są jednak łatwiej interpretowalne przez człowieka. Kolor jest wówczas definiowany przy użyciu parametrów odcienia, nasycenia i intensywności. Odcień określa częstotliwość (lub długość fali) dominującej składowej występującej w widmie, intensywność określa poziom jej energii, a nasycenie jest stosunkiem energii składowej dominującej do średniej energii pozostałych składowych w widmie i decyduje o „czystości", lub też „monochromatyczności" koloru.

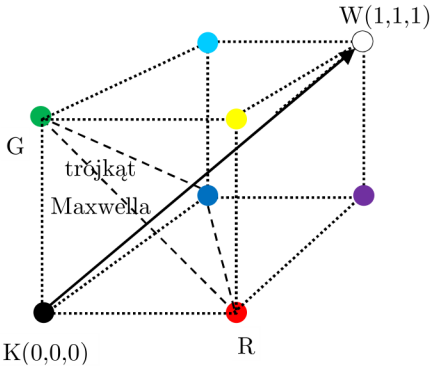
Obiekty na obrazach rzeczywistych, pochodzących z kamery, są rzadko oświetlone światłem idealnie rozproszonym, skutkiem czego ich piksele różnią się jasnością. Segmentacja obrazu powinna zatem korzystać z przestrzeni kolorów umożliwiającej uniezależnienie od jasności [16]. Światło białe wpływa na wartość każdego z komponentów RGB pikseli. Korekcję tych wartości można przeprowadzić normalizując kolor pikseli jego intensywnością. Takie współrzędne określa się mianem chromatycznych lub znormalizowanych

$$[r, g, b] = \frac{[R, G, B]}{R + G + B} \tag{2.1}$$

Spełnione jest wówczas równanie:

$$r + g + b \ = \ 1 \tag{2.2}$$

Jest to równanie płaszczyzny w przestrzeni RGB przechodzącej przez kolory R, G i B i wyznaczającej tzw. trójkąt Maxwella, na który rzutowane są środkowo wszystkie kolory z przestrzeni RGB (Rys. 10). Wektorem prostopadłym do trójkąta Maxwella jest wektor szarości od czerni do bieli. Ponieważ wszystkie współrzędne związane są równaniem (2.2), wystarczą dwie współrzędne do wyznaczenia diagramu chromatyczności (np. r, g).

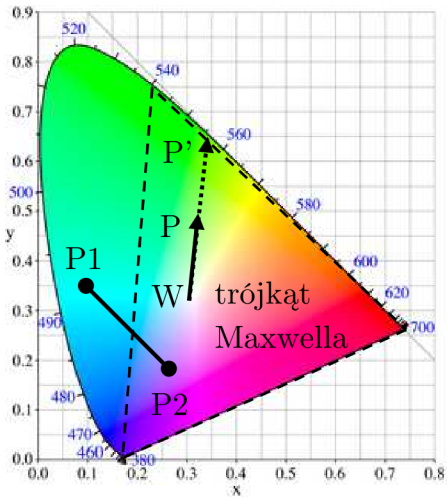


Rys. 10. Trójkąt Maxwella w przestrzeni RGB

Części kolorów z widzialnego spektrum nie można uzyskać na drodze addytywnego mieszania podstawowych barw R, G i B. Dodatkowo, zakres możliwych do uzyskania barw zależy od odcienia użytych barw podstawowych w urządzeniach rejestrujących i wyświetlających obrazy. Niezależną od urządzeń jest przestrzeń sRGB, zdefiniowana dla monitora standardowego (IEC 61966-2-1/FDIS, 1999). Międzynarodowa komisja do spraw oświetlenia CIE (Commision Internationale de l'Eclairage) już w 1931 roku ustanowiła standard CIE XYZ, oparty na trzech podstawowych kolorach X, Y i Z, które nie są realizowalne fizycznie, ale za pomocą których można, w wyniku mieszania addytywnego, uzyskać wszystkie kolory z zakresu widzialnego światła. Współrzędne chromatyczne tej przestrzeni mają zawsze w tym zakresie wartości nieujemne, a składowa Y określa luminancję koloru.

$$[x, y, z] = \frac{[X, Y, Z]}{X + Y + Z} \tag{2.3}$$

Podobnie jak dla przestrzeni RGB, można wyznaczyć diagram chromatyczny z pominięciem współrzędnej z, zależnej od dwóch pozostałych (Rys. 11).



Rys. 11. Diagram chromatyczności CIE

Na diagramie CIE XYZ możliwe są operacje wyznaczania koloru poprzez mieszanie - kolor wypadkowy leży na linii łączącej mieszane kolory lub w wielokącie przez nie utworzonym, jeśli mieszamy więcej niż dwie barwy. W związku z tym możliwe jest m.in. wyznaczanie barw dopełniających, które muszą leżeć na odcinku przechodzącym przez punkt bieli. Nasycenie koloru można zdefiniować jako stosunek długości odcinka od koloru od bieli (PW) w stosunku do długości odcinka przedłużonego do barwy maksymalnie nasyconej, leżącej na obwodzie diagramu, zwanego lokusem widma (P'W). Widać również, że nie można dokładnie pokryć całej gamy widzialnych kolorów poprzez mieszanie addytywne jedynie trójki kolorów podstawowych – tworzą one bowiem na diagramie zawsze pewien trójkąt. Model nie zachowuje równomierności percepcyjnej pomiędzy porównywanymi kolorami - odległość pomiędzy dwoma kolorami w przestrzeni nie odpowiada jednoznacznie różnicy postrzeganej przez człowieka. Prace Davida MacAdam i Richarda Huntera we wczesnych latach 40-tych ubiegłego wieku i późniejsze modyfikacje doprowadziły do stworzenia standardów CIELAB oraz CIELUV, które są modelami nieliniowymi, ale bardziej równomiernymi percepcyjnie [17].

Można zauważyć, że normalizacja parametrów RGB względem intensywności uniezależnia pomiar koloru od zmian intensywności, ale jednocześnie uniemożliwia rozpoznawanie czerni. Z tej przyczyny konstruowane są przestrzenie, w których kolory opisane są za pomocą niezależnych i intuicyjnych w interpretacji parametrów odcienia (hue), nasycenia (saturation) lub chromy i jasności lub intensywności (luminance). Najpopularniejszymi przestrzeniami kolorów z tej grupy są HSV, HSL i HSI. Różnią się one definicją parametru jasności i nasycenia. Konwersję z przestrzeni RGB przeprowadza się rzutując ortogonalnie kolory na płaszczyznę chromatyczną. Jasność koloru określamy jako odległość euklidesową od czerni V lub odległość od płaszczyzny chromatycznej (intensywność I). W modelu HSL wprowadza się pojęcie luminancji L zależnej od odległości od czerni i bieli.

Wartość maksymalną, środkową i minimalną z tripletu (r,g,b) oznaczmy jako:

$$M = \max(r, g, b),$$

$$m = \mathrm{med}(r, g, b),$$

$$u = \min(r, g, b),$$

wówczas:

$$V = \sqrt{M^2 + m^2 + u^2} \approx M \tag{2.4}$$

$$I = \frac{M + m + u}{3} \tag{2.5}$$

$$L = (\mathrm{M} + \mathrm{u})/2 \tag{2.6}$$

Nasycenie związane jest z kątem α pomiędzy wektorem jasności i linią szarości. Mimo jednoznacznego określenia chromy C jako odległości koloru od linii szarości, w teorii koloru często spotykamy się z problemem definicji parametru nasycenia. Dla omawianych przestrzeni:

$$S_{HSV} = C/V \tag{2.7}$$

$$S_{HSL} = \begin{cases} \dfrac{C}{2L} \ dla \ L \leq 1/2 \\ \dfrac{C}{2(1-L)} \ dla \ L > 1/2 \end{cases} \tag{2.8}$$

$$S_{HSI} = 1 - \frac{u}{I} \tag{2.9}$$

przy czym wartość nasycenia dla czerni przyjmuje się równą zeru. Błąd względny wyznaczenia nasycenia przy czerni dla omawianych modeli wynosi:

$$\delta S_{HSV} = \frac{\frac{C}{V} - \frac{C}{V+\varepsilon}}{\frac{C}{V}} = 1 - \frac{V}{V+\varepsilon} = \frac{\varepsilon}{V+\varepsilon} \tag{2.10}$$

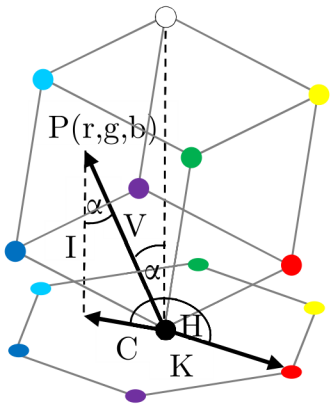
$$\delta S_{HSL} = \frac{\frac{C}{2L} - \frac{C}{2L+\varepsilon}}{\frac{C}{2L}} = 1 - \frac{2L}{2L+\varepsilon} = \frac{\varepsilon}{2L+\varepsilon} \tag{2.11}$$

$$\delta S_{HSI} = \frac{1 - \frac{u}{I} - \left(1 - \frac{u}{I+\varepsilon}\right)}{1 - \frac{u}{I}} = \frac{\varepsilon u}{(I+\varepsilon)(u-I)} \tag{2.12}$$

Z przytoczonych zależności wynika, że dla kolorów bliskich czerni ($I,V,L = 0$) parametr nasycenia obarczony jest dużym błędem (do 100%) i przyjmuje wartość szumową, podobnie jak odcień dla kolorów bliskich linii szarości.

W przyjętym modelu kolorów HCI (Hue, Chroma, Intensity) przestrzeń kolorów stanowi sześcian RGB obrócony tak, by linia szarości (od czerni do bieli) była prostopadła do płaszczyzny chromatycznej, na której współrzędne H i C są współrzędnymi biegunowymi rzutu ortogonalnego koloru na tę płaszczyznę (Rys. 12). W modelu HCI można zaniedbać jasność koloru, określającą udział światła białego, jeśli kolory pikseli mają dobrze określony odcień (mają dostatecznie dużą wartość chromy), i porównywać jedynie ich składową odcienia - Hue. Jeśli któryś z pikseli jest szary, największy wpływ na funkcję porównania ma różnica obu chrom. Jeśli oba są szare - porównywana jest ich intensywność [18].

Obiekty oświetlone bardzo mocno lub bardzo słabo mogą mieć kolor, który nie mieści się w założonej przestrzeni. Ich komponenty są wówczas ograniczane do przyjętego zakresu możliwych zmian. Można mówić wówczas o „prześwietleniu" kolorów, które zmieniają swoje nasycenie i odcień, dążąc do bieli lub czerni.



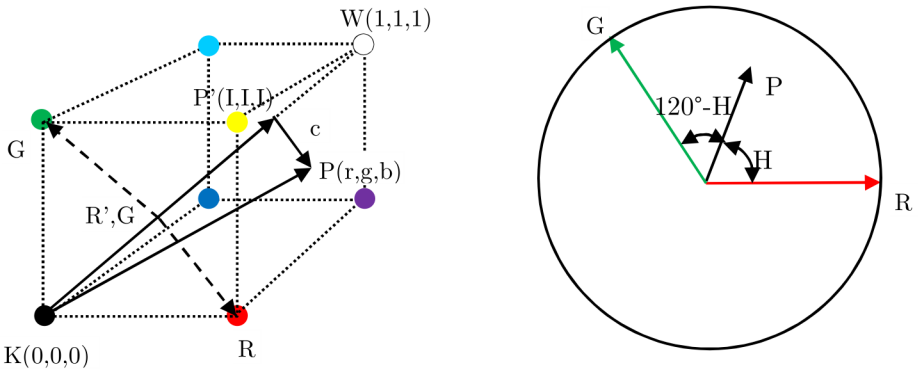
Rys. 12. Przestrzeń kolorów używana podczas segmentacji obrazów

Podczas segmentacji obrazu, do porównania kolorów, posłużono się wartością chromy a nie nasycenia, z następujących powodów:

- Przy jasności bliskiej poziomowi czerni, nasycenie definiowane jako stosunek chromy do jasności dąży do wartości maksymalnej (model HSV, HSL).
- Nie wszystkie obiekty są ciemne z powodu słabego oświetlenia – mogą to być obiekty ciemnoszare, pochłaniające promieniowanie w szerokim zakresie widma. Nasycenie koloru takiego obszaru obarczone jest największym błędem (HSV, HSL, HSI).
- Obrazy z kamery mają małe wartości chromy – w obecności szumów błąd wyznaczania nasycenia może być bardzo duży
- Dla pikseli jasnych i ciemnych chroma jest modyfikowana jasnością piksela w związku z efektem prześwietlenia

2.2 Model cylindryczny HCI

Konwersję pomiędzy przestrzeniami RGB i HCI najłatwiej przeprowadzić wyznaczając współrzędne biegunowe na płaszczyźnie chromatycznej, prostopadłej do przekątnej sześcianu RGB, przy użyciu normy euklidesowej odległości pomiędzy kolorami (Rys. 13). Przyjęto, że przedział zmienności komponentów R, G, B wynosi <0,1>. W cylindrycznym modelu HCI, wszystkie kolory oprócz podstawowych nie osiągają maksymalnych nasyceń. Wynika to stąd, że model RGB nie pokrywa całej gamy możliwych kolorów.



Rys. 13. Wyznaczenie współrzędnych cylindrycznego modelu HSV

Dla koloru P, jego rzut na przekątną achromatyczną ma wszystkie współrzędne równe jego intensywności P'(I,I,I). Wektory KW i P'P są prostopadłe, a więc ich iloczyn skalarny jest równy zeru:

$$\overrightarrow{KW} \circ \overrightarrow{P'P} = r - I + g - I + b - I = 0$$

Stąd intensywność koloru P

$$I = \frac{r+g+b}{3},$$

zgodna z równaniem (2.5). Chroma bezwzględna, to długość wektora P'P

$$c = \left|\overrightarrow{P'P}\right| = \sqrt{(r-I)^2 + (g-I)^2 + (b-I)^2} = \sqrt{r^2+g^2+b^2-3I^2}$$

$$c_{max} = c(1,0,0) = c(0,1,0) = c(0,0,1) = c(1,1,0) = c(1,0,1) = c(0,1,1) = \sqrt{\frac{2}{3}}$$

Znormalizowana do przedziału <0,1> wartość chromy to stosunek chromy bezwzględnej c do wartości maksymalnej chromy (uzyskiwanej dla kolorów R,G,B,C,M,Y). Wynosi zatem:

$$C = \frac{c}{c_{max}} = \sqrt{\frac{3(r^2+g^2+b^2)-(r+g+b)^2}{2}} = \sqrt{r(r-g)+g(g-b)+b(b-r)}$$

Wartość składowej Hue określona jest jako kąt pomiędzy wektorami R'R i P'P. Na podstawie ich iloczynu skalarnego można zapisać:

$$\overrightarrow{R'R} \circ \overrightarrow{P'P} = \left|\overrightarrow{R'R}\right|\left|\overrightarrow{P'P}\right| \cos H = R'R_x \cdot P'P_x + R'R_y \cdot P'P_y + R'R_z \cdot P'P_z$$

$$\cos H = \frac{\frac{2}{3}(r-I) - \frac{1}{3}(g-I) - \frac{1}{3}(b-I)}{c \cdot c_{max}} = \frac{2r-g-b}{2C}$$

Aby określić ćwiartkę, w której znajduje się kąt H, potrzebna będzie również wartość funkcji sinus. Można wyznaczyć ją z iloczynu skalarnego wektora P'P z wektorem G'G, bowiem:

$$\overrightarrow{G'G} \circ \overrightarrow{P'P} = \left|\overrightarrow{G'G}\right|\left|\overrightarrow{P'P}\right| \cos(120° - H) = G'G_x \cdot P'P_x + G'G_y \cdot P'P_y + G'G_z \cdot P'P_z$$

$$\cos(120° - H) = \frac{-\frac{1}{3}(r - I) + \frac{2}{3}(g - I) - \frac{1}{3}(b - I)}{c \cdot c_{max}} = \frac{2g - r - b}{2C}$$

A ponieważ:

$$\cos(120° - H) = \cos 120° \cos H + \sin 120° \sin H = -\frac{1}{2}\cos H + \frac{\sqrt{3}}{2}\sin H$$

możliwe jest wyznaczenie funkcji sinus składowej Hue:

$$\sin H = \frac{2}{\sqrt{3}}\left(\cos(120° - H) + \frac{1}{2}\cos H\right) = \frac{\sqrt{3}(g - b)}{2C}$$

Przyjmując $C_{rg} = r - g$ oraz $C_{rb} = r - b$, można zapisać:

$$\begin{cases} \cos H = \dfrac{C_{rb} + C_{rg}}{2C} = \dfrac{\alpha}{C} \\ \sin H = \sqrt{3}\dfrac{C_{rb} - C_{rg}}{2C} = \dfrac{\beta}{C} \end{cases}$$

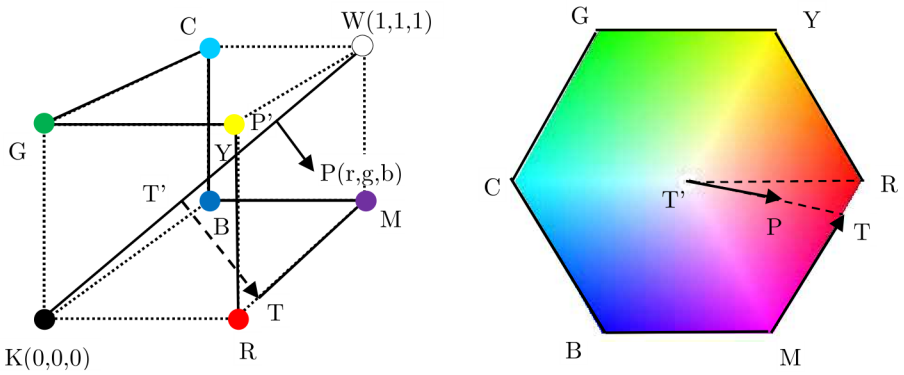
Zestawiając wszystkie parametry otrzymujemy sposób konwersji przestrzeni kolorów RGB na współrzędne biegunowe HCI (por. [16]):

$$\begin{cases} H = atan2(\beta, \alpha) \\ C = \sqrt{\alpha^2 + \beta^2} \\ I = (r + g + b)/3 \end{cases}, \text{ gdzie } \begin{cases} 2\alpha = C_{rb} + C_{rg} = 2r - g - b \\ 2\beta = \sqrt{3}(C_{rb} - C_{rg}) = \sqrt{3} \cdot (g - b) \end{cases} \tag{2.13}$$

2.3 Model heksagonalny HCI

Rzutem ortogonalnym sześcianu RGB w kierunku wektora szarości KW jest sześciokąt chromatyczny z informacją o chromie i odcieniu kolorów. Na obwodzie tego sześciokąta wszystkie kolory mają maksymalną chromę (i nasycenie). Odnajdując wektor T'T, będący wyskalowanym i przesuniętym wektorem P'P, którego początek znajduje się na linii szarości a koniec na krawędzi sześciokąta RYGCBM, chromę koloru P można wyrazić jako stosunek długości wektorów |P'P|/|T'T| (Rys. 14). Odcień Hue w modelu heksagonalnym określa iloraz długości obwodu h liczonego wzdłuż krawędzi sześciokąta RYGCBM od punktu R do punktu T i długości całego obwodu (równego sześciu).

$$C = \frac{\left|\overrightarrow{P'P}\right|}{\left|\overrightarrow{T'T}\right|}$$



Rys. 14. Wyznaczenie współrzędnych heksagonalnego modelu HCI

$$T = T' + \frac{\overrightarrow{P'P}}{C} = [I', I', I'] + \frac{\overrightarrow{P'P}}{C} = \left[I' + \frac{r-I}{C}, I' + \frac{g-I}{C}, I' + \frac{b-I}{C}\right]$$

Ponieważ T leży na krawędzi RYGCBM, to jedna z jego współrzędnych rgb jest równa 0 a druga wynosi 1. Trzecia współrzędna decyduje o nasyceniu i odcieniu koloru P:

$$T = \begin{cases} [1, T.g, 0], r > g > b \\ [T.r, 1{,}0], g > r > b \\ [0{,}1, T.b], g > b > r \\ [0, T.g, 1], b > g > r \\ [T.r, 0{,}1], b > r > g \\ [1{,}0, T.b], r > b > g \end{cases} \quad V' = \begin{cases} \frac{I-b}{C} \\ \frac{I-b}{C} \\ \frac{I-r}{C} \\ \frac{I-r}{C} \\ \frac{I-g}{C} \\ \frac{I-g}{C} \end{cases} \quad C = \begin{cases} r-b \\ g-b \\ g-r \\ b-r \\ b-g \\ r-g \end{cases} \quad h = \begin{cases} T.g = \frac{g-b}{C} \\ 2 - T.r = 2 - \frac{r-b}{C} \\ 2 + T.b = 2 + \frac{b-r}{C} \\ 4 - T.g = 4 - \frac{g-r}{C} \\ 4 + T.r = 4 + \frac{r-g}{C} \\ 6 - T.b = 6 - \frac{b-g}{C} \end{cases}$$

Oznaczmy wartość maksymalną, środkową i minimalną z tripletu (r,g,b) jako:

$$M = \max(r, g, b),$$

$$m = \mathrm{med}(r, g, b),$$

$$u = \min(r, g, b),$$

Wówczas odcień H w zakresie $< 0{,}360° >$ można zdefiniować jako wyskalowany do tego przedziału stosunek długości h do całego obwodu sześciokąta, równego 6:

$$H = 360 \cdot \frac{h}{6} = 60h = 60 \cdot \begin{cases} \frac{g-b}{C} + 0, dla\ r = \mathrm{M} \\ \frac{b-r}{C} + 2, dla\ g = \mathrm{M}, \\ \frac{r-g}{C} + 4, dla\ b = \mathrm{M} \end{cases} \tag{2.14}$$
$$H = H + 360, \qquad dla\ (H < 0)$$

Chromę można teraz określić krócej wyrażeniem:

$$C = \mathrm{M} - \mathrm{u} \tag{2.15}$$

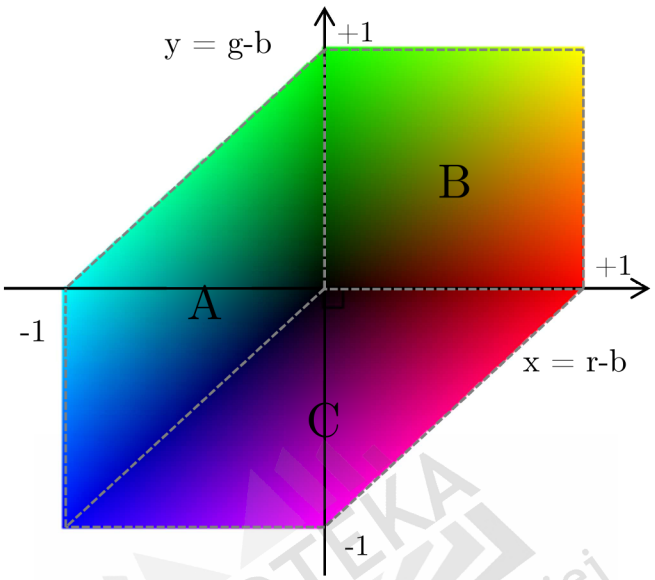
Nasycenie koloru w tym modelu nie jest normalizowane względem intensywności, Normalizacja taka wprowadza znaczne błędy przy małych intensywnościach koloru. Wraz z jego wzrostem, bezwzględna różnica pomiędzy wartościami komponentów (r,g,b) słabo nasyconego koloru może być jednak dużo większa niż przy czerni, dlatego normalizacja średnią jasnością obrazu może poprawić segmentację.

Intensywność jest określona tak jak poprzednio, jako średnia arytmetyczna komponentów (r,g,b), równaniem (2.5). Definicja intensywności, w odróżnieniu od jasności określonej równaniem (2.4), zapewnia podobny rozkład prawdopodobieństwa przyjmowanych wartości przez parametry opisujące model. Kolory w pełni nasycone nie przyjmują również maksymalnej intensywności. Dla przyjętych wartości HCI, komponenty (r,g,b) można wyznaczyć na podstawie zależności:

$$h = H/60$$
$$\begin{cases} r = I + \frac{C}{3}(2-h), b = r - C, g = 3I - r - b, dla\ 0 < h \leq 1 \\ g = I - \frac{C}{3}(0-h), b = g - C, r = 3I - g - b, dla\ 1 < h \leq 2 \\ g = I + \frac{C}{3}(4-h), r = g - C, b = 3I - r - g, dla\ 2 < h \leq 3 \\ b = I - \frac{C}{3}(2-h), r = b - C, g = 3I - r - b, dla\ 3 < h \leq 4 \\ b = I + \frac{C}{3}(6-h), g = b - C, r = 3I - g - b, dla\ 4 < h \leq 5 \\ r = I - \frac{C}{3}(4-h), g = r - C, b = 3I - r - g, dla\ 5 < h \leq 6 \end{cases} \tag{2.16}$$

Wyznaczenie wartości (r,g,b) na podstawie HCI umożliwia przeliczanie referencyjnego koloru dla segmentów, będącego średnią z wartości wszystkich kolorów pikseli segmentu w przestrzeni RGB, np. po modyfikacji parametru Hue filtracją statystyczną.

Z modelu heksagonalnego wynika, że odcień Hue koloru można w przybliżeniu przedstawić jako kąt na diagramie zależności sygnału różnicowego g-b w funkcji sygnału r-b. Diagram ten ma kształt obróconego i przeskalowanego sześciokąta chromatycznego ze zwiększonymi różnicami pomiędzy odcieniami zieleni i turkusu oraz purpury i czerwieni (Rys. 15)



Rys. 15. Heksagonalny diagram barw sygnałów różnicowych koloru

Po wyskalowaniu osi do przedziału zakresu przyjmowanych wartości przez komponenty RGB (255 dla składowych 8-bitowych) diagram zawiera wszystkie możliwe do uzyskania odcienie (kolory w pełni nasycone) dla dyskretnych wartości RGB.

Na podstawie współrzędnych (x,y) z zakresu <-1,1> łatwo wyznaczyć parametry (r,g,b) koloru w sektorach A, B i C:

$$\begin{cases} r=0 & g=y-x & b=-x & dla\ sektora\ A \\ b=0 & r=x & g=y & dla\ sektora\ B \\ g=0 & r=x-y & b=-y & dla\ sektora\ C \end{cases}$$

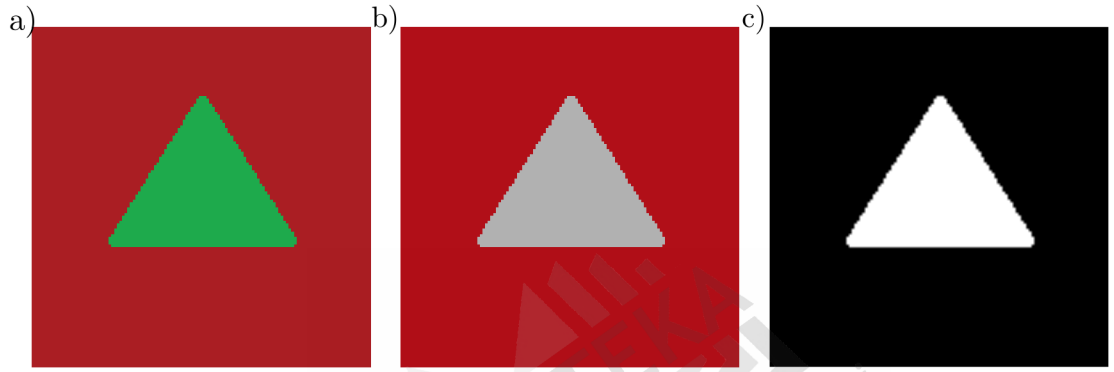
Odcień na diagramie można określić za pomocą funkcji atan2 wyznaczającej kąt w poprawnej ćwiartce na podstawie współrzędnych (x,y):

$$H \approx atan2(g-b, r-b) \tag{2.17}$$

2.4 Porównanie kolorów na podstawie parametrów HCI

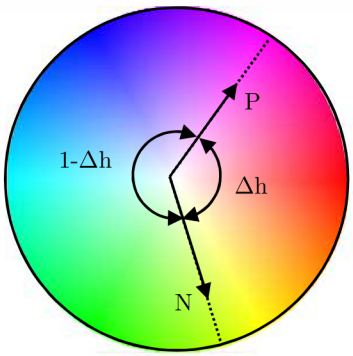
Aby uniezależnić segmentację obrazów kolorowych od intensywności oświetlenia światłem białym należy grupować piksele w oparciu o podobieństwo odcienia ich

koloru, np. w przestrzeni HCI. Jednak piksele szare, o zerowym nasyceniu mają nieokreślony odcień, gdyż proporcje ich składowych barw podstawowych są jednakowe. Piksele o małym nasyceniu wykazują dużą wrażliwość odcienia koloru od szumu w obrazie. Kąt dla pikseli położonych blisko środka koła chromatycznego może przyjmować wartości przypadkowe, związane z szumem lub przesunięciem punktu bieli na obrazie. Dla takich pikseli miarą podobieństwa może być różnica nasycenia i intensywności kolorów. Funkcja porównująca kolory musi zatem uwzględniać wszystkie trzy różnice składowych koloru: odcienia, nasycenia i intensywności (Rys. 16).



Rys. 16. Przykład segmentów różniących się wyłącznie: odcieniem (a), nasyceniem (b) i intensywnością (c)

Ponieważ miarą odcienia koloru jest kąt, różnica pomiędzy nimi sprowadza się do zakresu <-180°, 180°> i może być znormalizowana do przedziału <0,1>. Wartości z tego zakresu przyjmować będą również różnice w nasyceniu i jasności kolorów i łatwiejsze stanie się porównywanie kolorów o dowolnym nasyceniu (Rys. 17).



Rys. 17. Wyznaczenie różnicy odcieni pikseli kolorowych

Różnice pomiędzy odcieniami, nasyceniami i jasnościami kolorów P i N wynoszą:

$$\begin{cases} \Delta H = 2 * \min(\Delta h, 1 - \Delta h) \ , gdzie \ \Delta h = abs(P.H - N.H)/360 \\ \Delta C = abs(P.C - N.C) \\ \Delta I = abs(P.I - N.I) \end{cases} \tag{2.18}$$

Wartość różnicy pomiędzy kolorami P i N można przedstawić jako:

$$\Delta = \Delta H * w_H + \Delta C * w_C + \Delta I * w_I \tag{2.19}$$

gdzie w_H, w_C, w_I są wagami kombinacji liniowej różnic składowych.

Przy nasyceniu większym od progu *colorLevel* różnica między kolorami zdeterminowana jest wyłącznie różnicą odcieni pikseli P i N. Poniżej tego progu wpływ różnicy kolorów maleje z kwadratem, rośnie natomiast wpływ różnicy nasyceń i jasności. Dla nasycenia kolorów P i N równego zeru, gdy piksele są szare, do porównania wykorzystywana jest wyłącznie różnica jasności. Jeśli tylko jeden z kolorów jest szary, decydujący wpływ ma różnica nasyceń kolorów. Funkcje wag zależą od dwóch parametrów związanych z nasyceniami porównywanych kolorów i są opisane zależnościami:

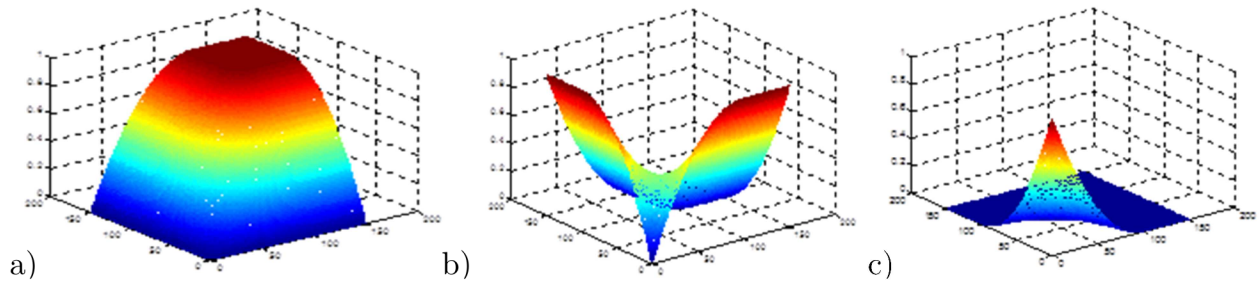
$$\begin{cases} w_H = (1-u)(1-v) \\ w_C = u(1-v) + v(1-u) \\ w_I = uv \end{cases} \tag{2.20}$$

$$u = \begin{cases} \left(1 - \dfrac{P.C}{colorLevel}\right)^2 & dla \ P.C < colorLevel \\ 0 & dla \ P.C \geq colorLevel \end{cases} \tag{2.21}$$

$$v = \begin{cases} \left(1 - \dfrac{N.C}{colorLevel}\right)^2 & dla \ N.C < colorLevel \\ 0 & dla \ N.C \geq colorLevel \end{cases} \tag{2.22}$$

Wagi określone zostały zatem jako funkcje dwuargumentowe, będące powierzchniami interpolacji biliniowej parametrów zależnych od kwadratu stosunku nasyceń do progu rozpoznania koloru *colorLevel*. Zależność kwadratowa umożliwia uzyskanie ciągłej pochodnej w punkcie nasycenia progowego colorLevel dla obu porównywanych kolorów. Wagi zmieniają się w przedziale <0,1> zależnie od nasyceń obu porównywanych kolorów. Oczywiście suma wag zawsze jest równa jedności:

$$w_H + w_C + w_I = 1$$

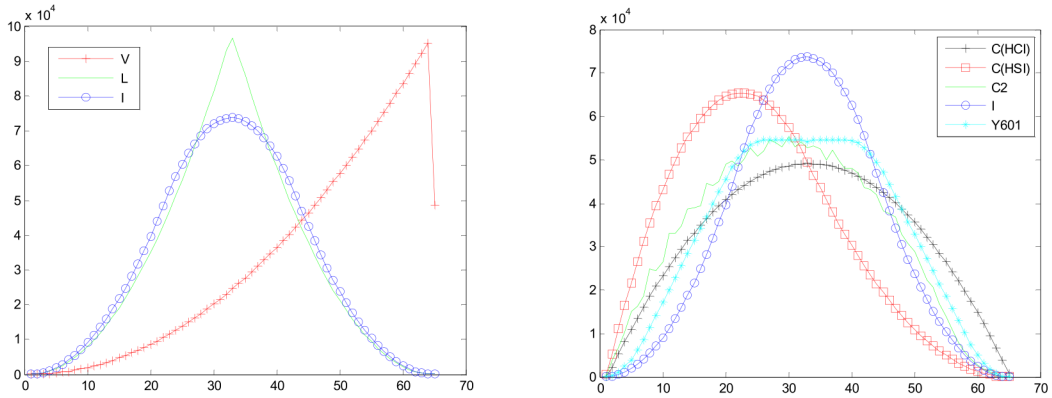


Rys. 18. Wagi różnicy odcieni w_H (a), nasycenia w_C (b) i intensywności w_I (c) funkcji porównującej kolory

Najważniejszą metodą klasy TSegment, odpowiadającą za porównanie z innym segmentem jest funkcja *DistanceTo*, oparta na porównaniu kolorów w przestrzeni HCI. Współrzędne HCI zostały określone na podstawie modelu heksagonalnego. Każda z metod segmentacji wykorzystuje tę metodę do określenia podobieństwa segmentów, dlatego jakość i szybkość segmentacji w dużym stopniu zależy od implementacji tej metody.

```
public double DistanceTo(TSegment seg)
{
    double colorLevel = Owner.ColorLevel;
    double dH = Hue - seg.Hue;
    if (dH < -0.5) dH += 1;
    else if (dH > 0.5) dH -= 1;
    dH = 2 * Math.Abs(dH);
    double dC = Math.Abs(Chroma - seg.Chroma);
    double dI = Math.Abs(Intensity - seg.Intensity);
    double u = Chroma / colorLevel; if (u > 1) u = 1;
    double v = seg.Chroma / colorLevel; if (v > 1) v = 1;
    u = (1 - u) * (1 - u);
    v = (1 - v) * (1 - v);
    dist = (1 - u)*(1 - v)*dH + (u*(1 - v) + v*(1 - u))*dC + u*v*dI;
    return dist;
}
```

Sposób wyznaczania różnic składowych kolorów wpływa oczywiście na ich rozkład statystyczny. Weźmy pod uwagę histogramy intensywności i chromy wyznaczone dla sześcianu RGB o 256 różnych wartościach r, g i b przy użyciu modeli HSV, HSL i HCI. Wartości jasności i nasycenia są uśredniane z czterech próbek, stąd ich zakres mieści się w przedziale od 0 do 64 (Rys. 19).



Rys. 19. Histogramy jasności i nasycenia przestrzeni kolorów HSV, HSL i HCI

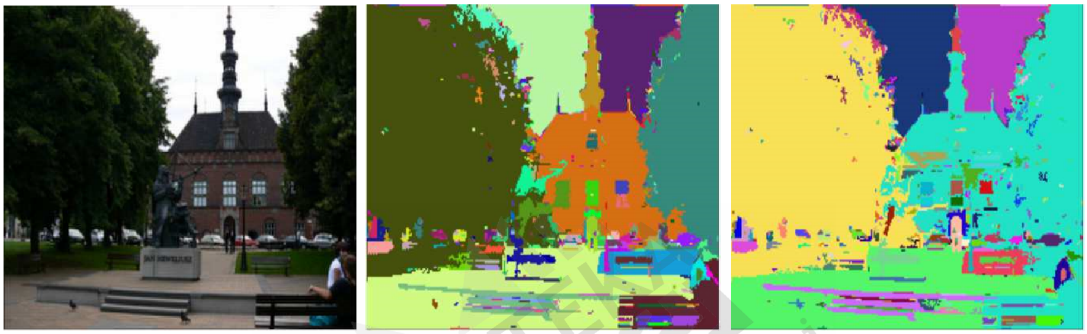
Jasności i nasycenia zostały wyznaczone według zależności:

$V = M;$
$L = (M + u) / 2;$
$I = (M + m + u) / 3;$
$Y_{601} = 0.3R + 0.59G + 0.11B$
$C(HCI) = M - u;$
$C(HSI) = (M + m - 2u) / 2;$
$C2 = \sqrt{r(r-g) + g(g-b) + b(b-r)}$

Histogramy nasycenia porównane zostały z histogramem wyznaczonym dla intensywności I i jasności Y_{601} (tzw. lumy), używanej do kodowania sygnału telewizji kolorowej i uwzględniającej wrażliwość oka ludzkiego. Z porównania widać, że w szczególności definicje C(HCI) i C2 dają dystrybucję wartości podobną do tych histogramów. Funkcja porównująca kolory jest kombinacją liniową różnic składowych odcienia, jasności i nasycenia, dlatego rozkłady wartości tych parametrów powinny być podobne. Warto zauważyć, że rozkład jasności V nie jest podobny do żadnej z podanych dystrybucji nasyceń.

Definicje C(HCI), I
Czas obliczeń:0.84s
Obiektów: 788

Definicje C2,V
Czas obliczeń:0.83s
Obiektów: 819



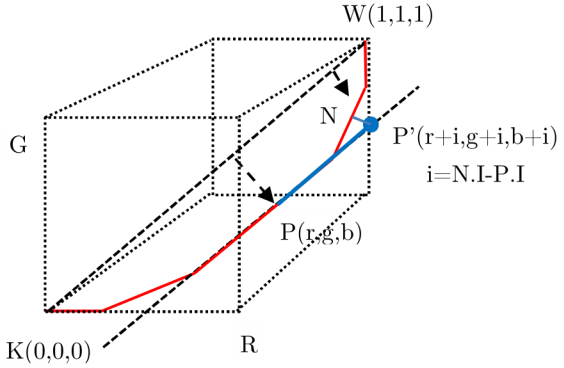
Definicje C(HCI), I
Czas obliczeń:0.83s
Obiektów: 435

Definicje C2,V
Czas obliczeń:0.88s
Obiektów: 499

Rys. 20. Segmentacja przy użyciu definicji nasycenia i jasności C(HCI), I oraz C2, V

2.5 Prześwietlenie kolorów

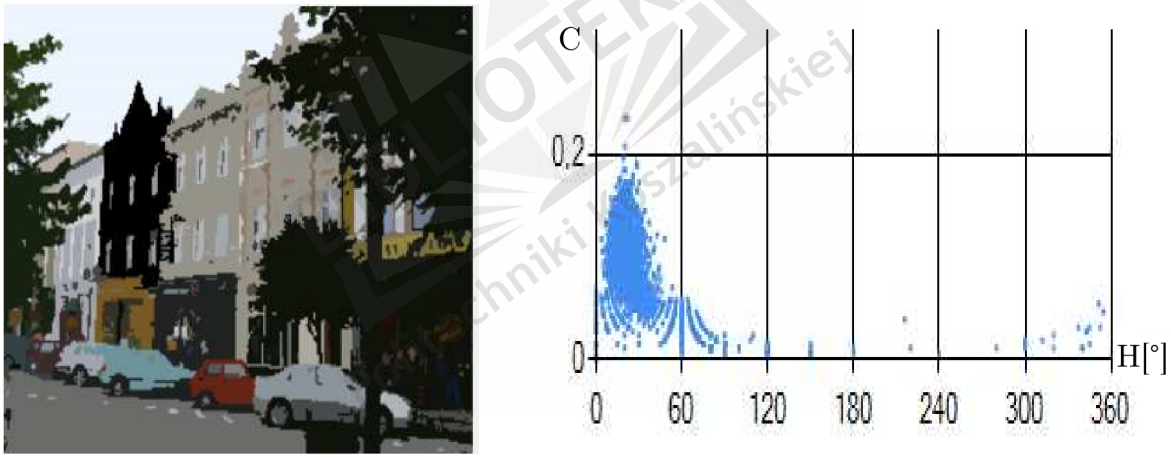
Przy zmianie intensywności koloru P jego składowe (r, g ,b) zmieniają się o ten sam współczynnik jasności *i*, przesuwając położenie koloru w przestrzeni RGB równolegle do wektora szarości. Jeśli kolor nie mieści się w sześcianie RGB, jego komponenty obcinane są przez detektor do wartości granicznej. Wpływa to na zmianę zarówno nasycenia jak i odcienia prześwietlonego koloru (Rys. 20).



Rys. 21. Wpływ zmiany intensywności na detekcję koloru

Dla zapisu 8-bitowego komponentów, zwiększenie jasności koloru P o współczynnik i spowoduje najpierw ograniczenie czynnika M do wartości 255 (dla $i > 255 - M$). Dla $i > 255 - m$ następuje również ograniczenie czynnika m. Prześwietlony kolor znajduje się wówczas dokładnie na krawędzi sześcianu RGB o odcieniu turkusowym, purpurowym lub żółtym. Przy zbyt słabym oświetleniu ograniczeniu podlega najpierw czynnik u a następnie m. Przy ograniczonym czynniku m, kolor ma dokładnie odcień czerwony, zielony lub niebieski (Rys. 22).
Nie można jednoznacznie stwierdzić, czy kolor zbyt jasny lub zbyt ciemny pochodzi od obiektu szarego, czy też jest to efekt prześwietlenia (lub niedoświetlenia). Podczas wyznaczania różnicy pomiędzy kolorami P i N można jednak porównać kolor N z prześwietlonym kolorem P' o tej samej jasności jak N. Jeśli kolor P' leży poza sześcianem RGB, jego składowe (r,g,b) należy ograniczyć do przedziału <0,255> (clamping):

$$P'(r,g,b) = P(r,g,b) + N.I - P.I$$
$$0 \leq P'(r,g,b) \leq 255 \tag{2.23}$$



Rys. 22. Przykładowy segment kamienicy (zaznaczony kolorem czarnym) oraz nasycenia wszystkich pikseli tego segmentu w funkcji odcienia. Wyraźnie widoczna jest zmiana odcienia dla małych nasyceń w kierunku wielokrotności 60° (krawędzi sześcianu RGB) w wyniku prześwietlenia i kwantyzacji

2.6 Podsumowanie

W rozdziale przedstawiono przestrzeń kolorów HCI zdefiniowaną przez parametry odcienia, chromy i intensywności koloru. Jej wybór podyktowany jest potrzebą zmniejszenia uzależnienia wyniku segmentacji od nierównomiernego oświetlenia obiektów. W przestrzeni ustanowiona została miara odległości, określona przez funkcję porównania kolorów. Dzięki wykorzystaniu parametru chromy, zamiast normalizowanego względem jasności nasycenia, możliwe stało się zdefiniowanie progu,

poniżej którego funkcja porównująca jest funkcją ważoną z różnic wszystkich trzech składowych HCI pomiędzy kolorami. Wagi dla różnic w odcieniu, chromie i intensywności są dwuargumentowymi funkcjami zależnymi od chromy jednego i drugiego koloru. Dla uzyskania gładkiej funkcji wykorzystano interpolację biliniową parametrów zależnych od kwadratu stosunku chromy koloru do zdefiniowanego progu koloru. Dla obu kolorów poniżej tego progu największy wpływ na odległość ma różnica intensywności kolorów, dla jednego koloru poniżej progu – różnica chrom. Dla obu kolorów powyżej progu, odległość pomiędzy nimi określona jest wyłącznie przez różnicę ich odcieni.

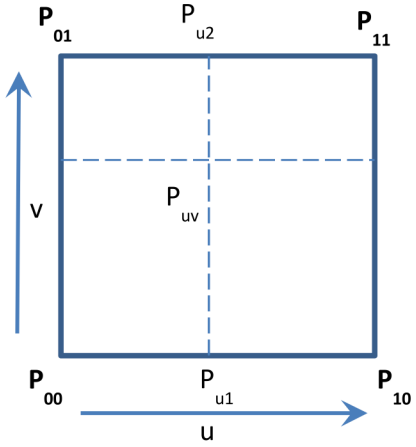
3. Przetwarzanie wstępne

3.1 Przeskalowanie obrazu źródłowego

Szybkość przetwarzania danych jest zależna od rozdzielczości przetwarzanego obrazu, dlatego obraz źródłowy powinien być przeskalowany do standardowych rozmiarów będących parametrem przetwarzania. Przystosowanie systemu do współpracy z różnymi rozdzielczościami kamer wymaga zamiany dyskretnej reprezentacji obrazu w postaci siatki pikseli na reprezentację ciągłą i ponowne próbkowanie (resampling) tak uzyskanego sygnału z nową częstotliwością próbkowania. Najczęściej stosowanymi algorytmami, biorącymi pod uwagę tylko najbliższe otoczenie piksela, jest interpolacja biliniowa lub bikubiczna. Reprezentację ciągłą uzyskujemy poprzez interpolację wartości pikseli pomiędzy węzłami siatki, lub w najprostszym przypadku – przez przyjęcie wartości najbliższego piksela (algorytm nearest neighbor). Interpolacja wykorzystywana jest również przy skalowaniu segmentów podlegających rozpoznaniu przy użyciu sieci SSN. Interpolacji podlegają próbki zarejestrowanych tonów instrumentów MIDI w celu uzyskania wszystkich dostępnych nut, wykorzystywanych do generowania dźwięku. Tony podlegają filtracji związanej z funkcją transmitancji ucha ludzkiego HRTF, która jest rejestrowana empirycznie dla dyskretnych wartości kątów azymutu i elewacji źródła dźwięku. Interpolacja biliniowa jest wykorzystywana w celu możliwości symulacji położenia dźwięku w dowolnej lokalizacji przestrzennej.

3.1.1 Interpolacja biliniowa

Jeśli wykorzystamy tylko cztery wartości w węzłach siatki, zakładając, że intensywności pikseli zmieniają się liniowo w kierunkach u i v pomiędzy tymi węzłami, można dokonać interpolacji biliniowej obrazu. Dla znormalizowanego przedziału $[0, 1]$ zmiennych u i v, wartość dowolnego piksela w kwadracie jednostkowym można uzyskać dzięki interpolacji liniowej węzłów w kierunku u, a następnie interpolacji liniowej tych wartości w kierunku v. Dla interpolacji biliniowej ten sam wynik uzyskamy interpolując w odwrotnej kolejności, najpierw w kierunku v, a następnie w kierunku u.



$$P_{u1} = (1-u)P_{00} + uP_{10}$$

$$P_{u2} = (1-u)P_{01} + uP_{11}$$

$$P_{uv} = (1-v)P_{u1} + vP_{u2}$$

Taki algorytm jest prosty w implementacji:

```
double Interpolate(double left, double right, double t)
{
    return left * (1 - t) + right * t;
}

double BilinearInterpolate(byte[,] P, double u, double v)
{
    double top = Interpolate(P[0, 0], P[1, 0], u);
    double bottom = Interpolate(P[0, 1], P[1, 1], u);
    return Interpolate(top, bottom, v);
}
```

Bardziej uniwersalny, ze względu na interpolację funkcjami sklejanymi wyższego stopnia jest zapis macierzowy (3.2), uzyskany z rozwinięcia (3.1):

$$P_{uv} = (1-u)(1-v)P_{00} + u(1-v)P_{10} + v(1-u)P_{01} + uvP_{11} \tag{3.1}$$

$$P_{uv} = \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix} = \boldsymbol{W_u P W_v}^{\boldsymbol{T}} \tag{3.2}$$

Alternatywnie płat powierzchni może być przedstawiony w postaci kanonicznej:

$$P_{uv} = \sum_{i=0}^{1}\sum_{j=0}^{1} w[i,j] * u^i v^j = w[0,0] + w[1,0]u + w[0,1]v + w[1,1]uv \tag{3.3}$$

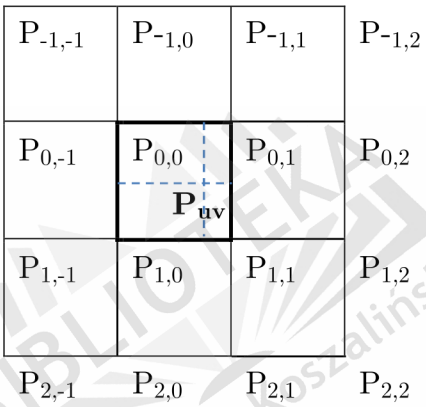
Na podstawie równania (3.1) można wyznaczyć macierz wag w:

$$w = \begin{bmatrix} P_{00} & P_{01} - P_{00} \\ P_{10} - P_{00} & P_{00} - P_{10} - P_{01} + P_{11} \end{bmatrix}$$

Interpolacja biliniowa wykorzystana została m.in podczas definicji funkcji porównującej kolory pikseli oraz do uzyskania ciągłej funkcji transmitancji HRTF ucha ludzkiego na pobudzenie akustyczne z dowolnego kierunku na podstawie siatki pomiarów wykonanych w projekcie IRCAM [19].

3.1.2 Interpolacja bikubiczna

Dla powierzchni opisanych wielomianami wyższego stopnia można zapewnić większą gładkość interpolacji biorąc pod uwagę również ciągłość pochodnych wyższego rzędu. W interpolacji bikubicznej uwzględniane są pochodne cząstkowe w kierunkach u i v oraz pochodne cząstkowe mieszane w każdym węźle. Ich wartość dla ciągłej pochodnej (zgodnie z uogólnionym twierdzeniem Schwarza) nie zależy od kolejności różniczkowania. Interpolacja bikubiczna nie zapewnia natomiast ciągłości drugich pochodnych w kierunkach u i v. Do wyznaczenia wartości pikseli w kwadracie jednostkowym [0,1]x[0,1] wykorzystywane jest 16 pikseli w węzłach siatki (Rys. 23). Sama metoda nie określa w jaki sposób wyznaczone mają być pochodne w węzłach kwadratu jednostkowego, można się w tym celu posłużyć np. metodą różnic skończonych lub krzywymi bikubicznymi (bicubic splines)



Rys. 23. Interpolacja bikubiczna danych w węzłach siatki

Płat powierzchni można przedstawić w postaci wielomianu:

$$P_{uv} = \sum_{i=0}^{3} \sum_{j=0}^{3} w[i,j] * u^i v^j \tag{3.4}$$

Wagi w (macierz 4x4) można wyznaczyć z 4 równań dla każdego z 4 węzłów $P_{x,y}$: $P_{0,0}$, $P_{0,1}$, $P_{1,0}$ i $P_{1,1}$ kwadratu jednostkowego, wykorzystując wartości powierzchni w węzłach i pochodnych cząstkowych w węzłach, określone na podstawie symetrycznego ilorazu różnicowego.

$$P_{x,y} = \sum_{i=0}^{3} \sum_{j=0}^{3} w[i,j] * x^i y^j$$

$$(P_{x,y})_u = \sum_{i=1}^{3} \sum_{j=0}^{3} w[i,j] * i x^{i-1} y^j \approx \frac{P_{x+1,y} - P_{x-1,y}}{2}$$

$$(P_{x,y})_v = \sum_{i=0}^{3} \sum_{j=1}^{3} w[i,j] * j x^i y^{j-1} \approx \frac{P_{x,y+1} - P_{x,y-1}}{2}$$

$$(P_{x,y})_{uv} = \sum_{i=1}^{3}\sum_{j=1}^{3} w[i,j] * ijx^{i-1}y^{j-1} \approx \frac{P_{x+1,y+1} - P_{x+1,y-1} - P_{x-1,y+1} + P_{x-1,y-1}}{4}$$

Zamieniając macierz w i dane w węzłach na jednowymiarowy wektor (układając wiersz za wierszem lub kolumna za kolumną), wyznaczenie wag sprowadza się do rozwiązania układu równań liniowych opisanych macierzą o rozmiarze 16x16 [20].

Płat powierzchni bikubicznej można również opisać za pomocą iloczynu tensorowego jednowymiarowych krzywych sklejanych - najczęściej krzywych Catmull-Rom'a. Procedura taka jest często opisywana w literaturze ze względu na wszechstronność zastosowań [21]. Zakładając jednak, że wielomian 1D jest kombinacją funkcji bazowych Bernsteina, a więc jest krzywą Beziera - zamiast założenia postaci kanonicznej, bardziej widoczne stają się symetrie warunków dla początku i końca krzywej. Łatwo również wyznaczyć wagi krzywej interpolacyjnej przechodzącej przez dwa węzły, o pochodnych w tych węzłach określonych przez dwa sąsiednie węzły. Potrzebne są zatem 4 węzły, oznaczone tu jako P_{-1}, P_0, P_1, P_2. Przyjmując pomocniczą zmienną $s = 1 - t$ i punkty kontrolne krzywej A_i, dla których $A_0 = P_0$ i $A_3 = P_1$, można zapisać:

$$P_t = \sum_{i=0}^{3} \binom{3}{i} s^{3-i}t^i A_i = s^3P_0 + 3s^2tA_1 + 3st^2A_2 + t^3P_1 \tag{3.5}$$

$$(P_t)_t = -3s^2P_0 + 3s^2A_1 - 6stA_1 + 6stA_2 - 3t^2A_2 + 3t^2P_1$$

Pochodna krzywej w węzłach P_0, P_1:

$$(P_0)_t = -3P_0 + 3A_1 \approx \frac{P_1 - P_{-1}}{2} \Rightarrow A_1 = P_0 + \frac{-P_{-1} + P_1}{6}$$

$$(P_1)_t = -3A_2 + 3P_1 \approx \frac{P_2 - P_0}{2} \Rightarrow A_2 = P_1 + \frac{P_0 - P_2}{6}$$

Krzywą pomiędzy węzłami P_0 i P_1, dla parametru t z przedziału [0,1] można teraz określić na podstawie wszystkich 4 węzłów:

$$P_t = [t^3 \quad 3st^2 \quad 3s^2t \quad s^3] \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1/6 & 1 & -1/6 \\ -1/6 & 1 & 1/6 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_{-1} \\ P_0 \\ P_1 \\ P_2 \end{bmatrix} = W \begin{bmatrix} P_{-1} \\ P_0 \\ P_1 \\ P_2 \end{bmatrix}$$

$$W[0] = -s^2t/2$$

$$W[3] = -st^2/2$$

$$W[1] = s^3 - W[3] - 6W[0] = s + W[3] - 2W[0]$$

$$W[2] = t^3 - W[0] - 6W[3] = t + W[0] - 2W[3]$$

Warto zauważyć, że $\sum_i W[i] = 1$.

Dla powierzchni można przeprowadzić interpolację najpierw w kierunku u, a następnie w kierunku v:

$$P_{uv} = W_u \begin{bmatrix} P_{-1,-1} & P_{-1,0} & P_{-1,1} & P_{-1,2} \\ P_{0,-1} & P_{0,0} & P_{0,1} & P_{0,2} \\ P_{1,-1} & P_{1,0} & P_{1,1} & P_{1,2} \\ P_{2,-1} & P_{2,0} & P_{2,1} & P_{2,2} \end{bmatrix} {W_v}^T = \sum_{i=0}^{3} \sum_{j=0}^{3} W_u[i] W_v[j] P_{i-1,j-1} \quad (3.6)$$

Podczas przesuwania maski węzłów względem obrazu wyznaczane są płaty powierzchni Beziera o takich samych pochodnych cząstkowych, które razem tworzą bikubiczną powierzchnię sklejaną.

Prosta implementacja interpolacji biliniowej i bikubicznej na podstawie równań (3.2) i (3.6) w C# może wyglądać następująco:

```
public double[] GetWeights(double t)
{
    double s = 1 - t;
    double[] result = new double[Bicubic ? 4 : 2];
    if (Bicubic)
    {
        result[0] = -t * s * s / 2;
        result[3] = -t * t * s / 2;
        result[1] = s + result[3] - 2 * result[0];
        result[2] = t + result[0] - 2 * result[3];
    }
    else
    {
        result[0] = s;
        result[1] = t;
    }
    return result;
}

double BiInterpolate(byte[,] P, double u, double v)
{
    double[] U = GetWeights(u);
    double[] V = GetWeights(v);
    double result = 0;
    for (int i = 0; i < U.Length; i++)
        for (int j = 0; j < V.Length; j++)
            result += U[i] * P[i, j] * V[j];
    return result;
}
```

Interpolacja biliniowa i bikubiczna zakłada parametryzację równomierną zmiennych u i v, niezależną od dynamiki zmienności powierzchni interpolowanego obrazu. Taką zależność można uzyskać stosując np. interpolację za pomocą płatów powierzchni B-sklejanych, wykorzystując wektory węzłów (miejsc połączeń krzywych) w kierunkach u i v.

3.1.3 Krzywe B-sklejane

Krzywe B-sklejane są kombinacją liniową funkcji bazowych N_i^m, określonych parametrycznie za pomocą wektora węzłów t^* o rozmiarze $N + m + 1$. Współczynnikami kombinacji c_i są punkty kontrolne nazywane punktami de Boora.

$$P(t) = \sum_{i=1}^{N} c_i \mathrm{N}_i^m(t) \tag{3.7}$$

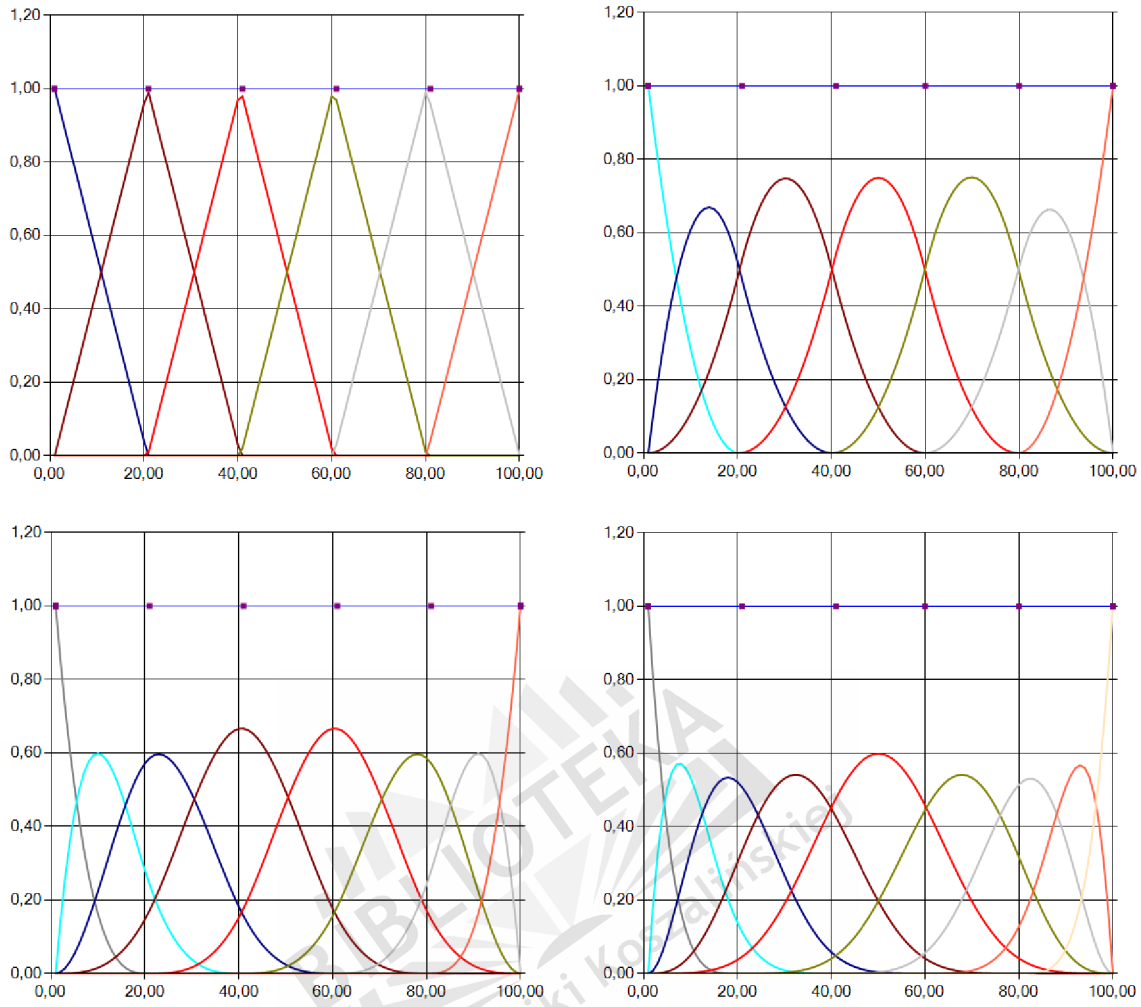
Funkcje bazowe są również krzywymi B-sklejanymi, dlatego ich naturalną definicją jest następująca zależność rekurencyjna [22]:

$$\begin{gathered} \mathrm{N}_i^0 = 1 \\ \mathrm{N}_i^m = \alpha_i \mathrm{N}_i^{m-1} + (1 - \alpha_{i+1}) \mathrm{N}_{i+1}^{m-1} \end{gathered} \tag{3.8}$$

Gdzie α jest funkcją liniową parametru t:

$$\alpha_i = \frac{t - t_i^*}{t_{i+m}^* - t_i^*}$$

Tym samym m jest stopniem wielomianów tworzących funkcje bazowe. Na Rys. 24 przedstawiono funkcje bazowe dla stopnia $m = 1..4$.



Rys. 24. Funkcje bazowe N_i^m krzywych B-sklejanych dla stopnia m = 1, 2, 3 i 4 (parametryzacja równomierna)

Funkcje bazowe mają nośnik zwarty i ograniczony do zakresu $t \in [t_i^*, t_{i+m+1}^*)$, tzn.

$$\sum_{i=1}^{N} N_i^m = 1$$

$$N_i^m = 0 \; dla \; t \notin [t_i^*, t_{i+m+1}^*)$$

Własność zwartości nośnika określa lokalny wpływ położenia punktu de Boora c_i na krzywą w przedziale $[t_i^*, t_{i+m+1}^*)$. Oznacza to również, że położenie punktu na krzywej zależy jedynie od *m+1* punktów de Boora. Ze względu na ograniczenie nośnika funkcji bazowych krzywa musi leżeć w sumie kolejnych powłok wypukłych *m+1* punktów kontrolnych de Boora. Wyznaczanie położenia kolejnych punktów na krzywej poprzez sumowanie *m+1* funkcji bazowych, np. przy rysowaniu krzywej, realizowane jest przez algorytm de Boora.

Zakładając, że krzywa jest opisana poprzez stopień wielomianu $m = Degree$ oraz wektor węzłów *Knots*, metoda implementująca w klasie C# wyznaczanie wartości funkcji bazowych N_i^m może być bardzo prosta, jeśli wykorzystuje rekurencję:

```
public double Base(int i, double t)
{
    double result = 0;
    if (Knots[i] == t || Knots[i] < t && t < Knots[i + Degree + 1])
    {
        if (Degree == 0)
            result = 1;
        else
        {
            double u0 = Knots[i + Degree] - Knots[i];
            double u1 = Knots[i + Degree + 1] - Knots[i + 1];
            if (u0 != 0) u0 = (t - Knots[i]) / u0;
            if (u1 != 0) u1 = (t - Knots[i + 1]) / u1;
            Degree--;
            result = Base(i, t) * u0 + Base(i + 1, t) * (1 - u1);
            Degree++;
        }
    }
    return result;
}
```

Interpolacja krzywymi B-sklejanymi sprowadza się do rozwiązania układu równań liniowych w postaci macierzy m-diagonalnej i wyznaczenia punktów kontrolnych c_i.

$$\begin{bmatrix} \mathrm{N}_1^m(t_1) & \dots & \mathrm{N}_N^m(t_1) \\ \dots & \dots & \dots \\ \mathrm{N}_1^m(t_N) & \dots & \mathrm{N}_N^m(t_N) \end{bmatrix} \begin{bmatrix} c_1 \\ \dots \\ c_N \end{bmatrix} = \begin{bmatrix} P(t_1) \\ \dots \\ P(t_N) \end{bmatrix} \tag{3.9}$$

Chcąc zapewnić stałą pochodną na końcach krzywej (przejście w linię prostą, druga pochodna równa zeru) należy do układu równań dodać warunki brzegowe, zwane warunkami Neumanna lub naturalnymi, konstruując tzw. krzywe normalne.

$$\begin{bmatrix} \mathrm{N}''^m_1(0) & \dots & \mathrm{N}''^m_N(0) \\ \mathrm{N}_1^m(t_1) & \dots & \mathrm{N}_N^m(t_1) \\ \dots & \dots & \dots \\ \mathrm{N}_1^m(t_n) & \dots & \mathrm{N}_N^m(t_N) \\ \mathrm{N}''^m_1(1) & \dots & \mathrm{N}''^m_N(1) \end{bmatrix} \begin{bmatrix} c_1 \\ \dots \\ c_N \end{bmatrix} = \begin{bmatrix} 0 \\ P(t_1) \\ \dots \\ P(t_N) \\ 0 \end{bmatrix} \tag{3.10}$$

O przebiegu krzywej w dużym stopniu decyduje parametryzacja, czyli rozkład węzłów, będących miejscem połączenia krzywych sklejanych. W węźle krzywa B-sklejana ma przeważnie ciągłość klasy C^{m-1}, tzn. wszystkie pochodne rzędu mniejszego od *m-1* są ciągłe. Dla k takich samych węzłów, ciągłość krzywej zmniejsza się o *k-1* i krzywa w węźle jest klasy C^{m-k}. Taki węzeł nazywamy k-krotnym. Wstawiając nowy węzeł lub zwiększając krotność istniejącego węzła, można dokładniej modyfikować krzywą sklejaną. Do rozmieszczenia węzłów stosowane są różne strategie [23] [24]:

– parametryzacja równomierna (uniform) rozmieszcza węzły w równych odstępach, niezależnie od sposobu wyznaczania współrzędnych punktów krzywej na podstawie parametru t:

$$s_i = \frac{i}{n} \tag{3.11}$$

– parametryzacja łukowa (chordal) – odległości pomiędzy węzłami są proporcjonalne do długości krzywej pomiędzy punktami określonymi przez równooddalone wartości parametru t. W praktyce przybliża się długość krzywej odcinkami prostymi pomiędzy punktami krzywej $P(t_j)$:

$$s_i = \frac{\sum_{j=1}^{i} \|P(t_j) - P(t_{j-1})\|}{\sum_{j=1}^{n} \|P(t_j) - P(t_{j-1})\|} \tag{3.12}$$

– parametryzacja centripetal – zdefiniowana podobnie jak łukowa, ale miarą odległości jest pierwiastek kwadratowy z odległości euklidesowej. Przy interpolacji nie tworzy pętli ani poduszek (loops & cusps) przebiegając bliżej łamanej łączącej punkty węzłowe.

$$s_i = \frac{\sum_{j=1}^{i} \sqrt{\|P(t_j) - P(t_{j-1})\|}}{\sum_{j=1}^{n} \sqrt{\|P(t_j) - P(t_{j-1})\|}} \tag{3.13}$$

Z punktu widzenia interpolacji i aproksymacji korzystniejsza jest oczywiście parametryzacja łukowa lub centripetal. Nie są jednak one niezależne od skalowania krzywej (not scale invariant). Proponowana parametryzacja używa normy Manhattan do wyznaczenia przybliżonej długości znormalizowanych odcinków krzywej oddzielnie w kierunkach X i Y, co pozwala uzyskać niezależność od skalowania.

$$s_i = 0.5 * \frac{\sum_{j=1}^{i} |P(x_j) - P(x_{j-1})|}{\sum_{j=1}^{n} |P(x_j) - P(x_{j-1})|} + 0.5 * \frac{\sum_{j=1}^{i} |P(y_j) - P(y_{j-1})|}{\sum_{j=1}^{n} |P(y_j) - P(y_{j-1})|} \tag{3.14}$$

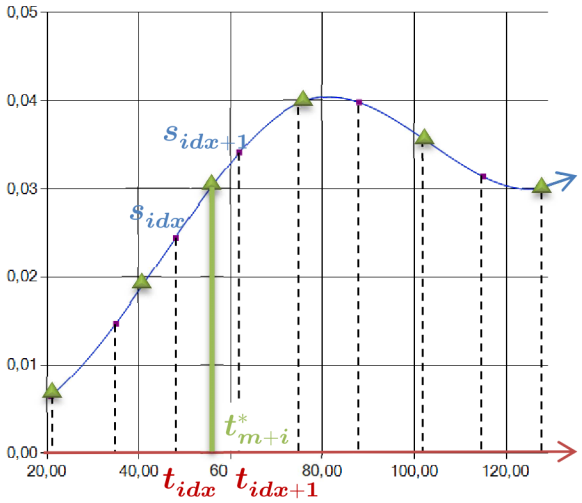
Na podstawie parametrów s, określane są położenia t_i^* węzłów krzywej. Węzły $t_0^* = t_1^* = \cdots = t_m^* = t_0$ oraz $t_n^* = t_{n+1}^* = \cdots = t_{n+m}^* = t_n$ są węzłami brzegowymi. Węzły wewnętrzne, dla $i = 1..i_{max}$, $i_{max} = n - m$ można wyznaczyć na podstawie interpolacji liniowej

$$t_{m+i}^* = t_{idx} + \frac{i/i_{max} - s_{idx}}{s_{idx+1} - s_{idx}} (t_{idx+1} - t_{idx}) \tag{3.15}$$

Gdzie *idx* jest indeksem spełniającym nierówność:

$$s_{idx} < i/i_{max} < s_{idx+1}$$

Taki sposób wyznaczania węzłów ilustruje Rys. 25.



Rys. 25. Interpolacja liniowa węzłów t* na podstawie parametrów s

Standardowym podejściem przy wyznaczaniu węzłów wewnętrznych jest jednak parametryzacja równomierna lub uśrednianie m kolejnych parametrów wektora s

$$t^*_{m+i} = \frac{1}{m} \sum_{j=i}^{i+m-1} s_j \tag{3.16}$$

Dzięki uśrednianiu układ równań interpolacyjnych jest zdefiniowany dodatnio i zawsze posiada rozwiązanie. W przypadku interpolacji położenia węzłów, konstruowana macierz może być nieodwracalna, co związane jest z niespełnieniem warunku Schoenberga-Whitney [25] zawierania się co najmniej jednego parametru t_i pomiędzy węzłami t^*_i *i* t^*_{i+m+1}.

$$t^*_i \leq \ t_i \leq t^*_{i+m+1}$$

Warunek ten można również odwrócić na ograniczenie położenia węzłów (szczególnie przydatne przy aproksymacji, kiedy liczba węzłów jest mniejsza od liczby parametrów)

$$t_{i-m-1} \leq t^*_i \leq \ t_i$$

Warunek można sprawdzić i przeprowadzić korekcję położenia węzłów do żądanego przedziału w momencie wyznaczania wartości węzłów, zapewniając konstrukcję macierzy odwracalnej.

Łatwo zauważyć, że funkcje bazowe kolejnych stopni można wyznaczyć poprzez konwolucję prostokątnych impulsów, będących funkcjami bazowymi zerowego stopnia [26].

$$\mathrm{N}^m_i = \mathrm{N}^0_i * \mathrm{N}^0_{i+1} * \ldots * \mathrm{N}^0_{i+m} \tag{3.17}$$

Dla parametryzacji równomiernej, kolejne funkcje bazowe tego samego stopnia są przesuniętymi wersjami tej samej funkcji. Widmo Fouriera funkcji bazowej

budowanej z symetrycznych funkcji impulsowych wokół centralnej funkcji m-tego stopnia ma postać:

$$\widehat{\mathrm{N}}_0^m(\omega) = sinc^{m+1}(\omega/2) \tag{3.18}$$

Jawna postać funkcji bazowej stopnia m w dziedzinie przestrzeni jest wówczas określona przez równanie:

$$\mathrm{N}_0^m(t) = \frac{1}{m!}\sum_{k=0}^{m+1}\binom{m+1}{k}(-1)^k\left(t-k+\frac{m+1}{2}\right)_+^m \tag{3.19}$$

przy czym indeks „+" oznacza, że dla wartości ujemnych funkcja przyjmuje wartość zerową.

Przy *m+1*-ukrotnieniu węzła, krzywa przechodzi przez punkt de Boora. Dla szczególnego przypadku dwóch kolejnych takich węzłów, funkcje bazowe stają się wielomianami Bernsteina

$$\mathrm{N}_i^m = \binom{m}{i}(1-t)^{m-i}t^i \tag{3.20}$$

Krzywe B-sklejane są zatem uogólnieniem krzywych Beziera. Dużą popularność w grafice zyskały ważone krzywe i powierzchnie B-sklejane o niejednorodnej parametryzacji – NURBS (Non-Uniform Rational B-Splines) [27]. Dzięki dodaniu wag do funkcji bazowych są one jeszcze szerszą klasą krzywych i powierzchni sklejanych, umożliwiającą modelowanie wszystkich przekrojów stożka (elipsy, hiperbole, parabole). Krzywe NURBS opisane są równaniem:

$$P(t) = \frac{\sum_{i=1}^{N} c_i \mathrm{N}_i^m(t) w_i}{\sum_{i=1}^{N} \mathrm{N}_i^m(t) w_i} \tag{3.21}$$

3.1.4 Powierzchnie B-sklejane

Powierzchnie można interpolować krzywymi B-sklejanymi oddzielnie w kierunku u i v. Dla krzywych stopnia wyższego od 1 rozwiązanie zależy od kolejności wyboru kierunku interpolacji. Powierzchnię B-sklejaną można również utworzyć przy użyciu iloczynu tensorowego krzywych B-sklejanych, zakładając, że powierzchnia tworzona jest na podstawie funkcji bazowych w kierunku u, a punkty de Boora układają się na krzywej B-sklejanej w kierunku v. Powierzchnia utworzona z funkcji bazowych w kierunku v, o punktach de Boora na krzywej w kierunku u opisuje tę samą powierzchnię.

$$P(u,v) = \sum_{i=1}^{N} c_i(v)\mathrm{N}_i^m(u) = \sum_{i=1}^{N}\sum_{j=1}^{M} c_{i,j}\mathrm{N}_j^m(v)\mathrm{N}_i^m(u) \tag{3.22}$$

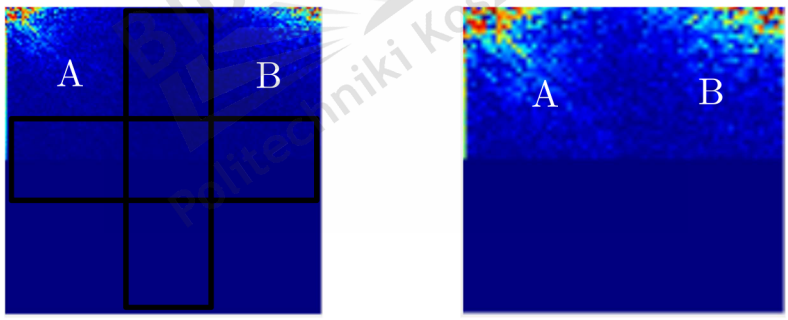
3.1.5 Transformacja Fouriera 2D

Zmianę rozdzielczości obrazu można przeprowadzić wykorzystując dwuwymiarową transformację Fouriera (*fft2*). Algorytm FFT2 wymaga rozdzielenia kanałów R, G i B do trzech tablic z intensywnościami składowych koloru. Każdy kanał jest przetwarzany w ten sam sposób. W proponowanym rozwiązaniu użyta została transformacja Fouriera z wykorzystaniem biblioteki *fftw* (Fastest Fourier Transform in the West) rozwijanej w instytucie MIT (Massachussets Institute of Technology). Aby móc wykorzystać bibliotekę w środowisku C#, stworzony został wrapper w postaci statycznej klasy FFTW z metodami transformacji prostej *fft2* i odwrotnej *ifft2* obrazu 2D oraz *fft* i *ifft* dla sygnałów 1D. Tym samym sposób posługiwania się transformacjami jest podobny jak w środowisku MATLAB. Ze względu na rzeczywiste wartości intensywności pikseli, informacja o widmie zawarta jest w 4 ćwiartkach sprzężonych parami. Metody transformacji wykorzystują symetrię widma częstotliwościowego obrazu i przetwarzają jedynie dwie ćwiartki.

```
public static complex[] fft(double[] input)
{
    int N = input.Length;
    complex[] output = new complex[N / 2 + 1];
    fixed (double* inPtr = input)
    fixed (complex* outPtr = output)
    {
        IntPtr FFT = fftw_plan_dft_r2c_1d(N, inPtr, outPtr, flags.Estimate);
        fftw_execute(FFT);
        fftw_destroy_plan(FFT);
    }
    return output;
}

public static double[] ifft(complex[] input)
{
    int N = 2 * (input.Length - 1);
    double[] output = new double[N];
    fixed (complex* inPtr = input)
    fixed (double* outPtr = output)
    {
        IntPtr FFT = fftw_plan_dft_c2r_1d(N, inPtr, outPtr, flags.Estimate);
        fftw_execute(FFT);
        fftw_destroy_plan(FFT);
    }
    return output;
}

public static complex[,] fft2(double[,] input)
{
    int W = input.GetLength(0);
    int H = input.GetLength(1);
    complex[,] output = new complex[W, H / 2 + 1];
    fixed (double* inPtr = input)
    fixed (complex* outPtr = output)
    {
        IntPtr FFT2 = fftw_plan_dft_r2c_2d(W, H, inPtr, outPtr, flags.Estimate);
        fftw_execute(FFT2);
        fftw_destroy_plan(FFT2);
    }
```

```
    return output;
}

public static double[,] ifft2(complex[,] input)
{
    int W = input.GetLength(0);
    int H = 2 * (input.GetLength(1) - 1);
    double[,] output = new double[W, H];
    fixed (complex* inPtr = input)
    fixed (double* outPtr = output)
    {
        IntPtr IFFT2 = fftw_plan_dft_c2r_2d(W, H, inPtr, outPtr, flags.Estimate);
        fftw_execute(IFFT2);
        fftw_destroy_plan(IFFT2);
    }
    return output;
}
```

Ponieważ nie jest wykonywana operacja przesunięcia widma (*fftshift*), najwyższe częstotliwości znajdują się w jego środku. Najwyższe częstotliwości poziome (x) i pionowe (y) są eliminowane, co redukuje jednocześnie informacje związane z szybkozmiennymi harmonicznymi sygnału. Pozostała informacja jest kopiowana do nowego okna o rozmiarze pomniejszonego obrazu. Obie ćwiartki A i B przetwarzane są jednocześnie w tej samej pętli. W każdej iteracji kopiowany jest i-ty piksel od początku wiersza ćwiartki A i i-ty piksel od końca wiersza ćwiartki B (Rys. 26). Z tak zmodyfikowanego widma odtwarzany jest obraz o pomniejszonej rozdzielczości.



Rys. 26. Zmniejszenie rozdzielczości obrazu (aproksymacja) przez wycięcie części widma wysokoczęstotliwościowego

Obliczenia związane z interpolacją wysokiej jakości mogą okazać się bardziej kosztowne od przetwarzania obrazu po niewielkiej redukcji rozdzielczości metodą prostszą obliczeniowo. Na przykład zmiana rozmiaru kolorowego obrazu wejściowego za pomocą transformacji Fouriera wymaga wykonania sześciu transformacji FFT 2D (trzech prostych i trzech odwrotnych dla każdego kanału R, G i B). Sposób przeskalowania może mieć duże znaczenie dla poprawności segmentacji obrazu, dlatego domyślnym algorytmem jest jednak filtracja dolnoprzepustowa wykorzystująca transformację Fouriera.

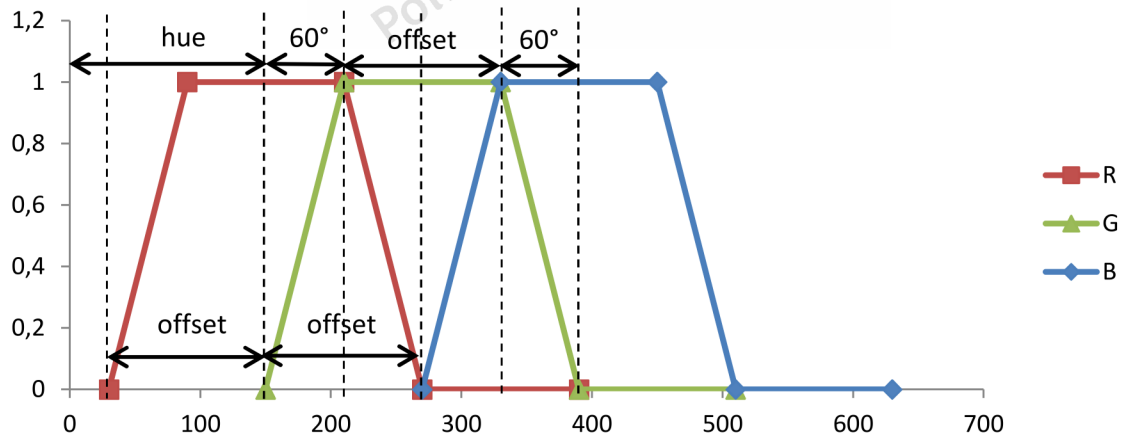
3.2 Pseudokolory

Wynik segmentacji obrazu można przedstawić w postaci obrazów pseudokolorowych indeksowanych. Przetwarzanie obrazów za pomocą klasy *Bitmap* w środowisku .NET4.0 może być jednak bardzo nieefektywne. Zwłaszcza, gdy wykorzystywane są metody *GetPixel* i *SetPixel* do pobierania i modyfikacji wartości pojedynczych pikseli. W celu poprawy wydajności tych operacji stworzona została specjalna klasa *Pixelmap*, która zawiera tablicę 32-bitowych wartości typu integer. Tablica *Pixels* określa wartości poszczególnych pikseli w obrazach RGB (8 bitów na komponent) lub indeksy do palety kolorów. Klasa umożliwia tym samym pracę z obrazami RGB i indeksowanymi. Zawiera tablicę kolorów *Palette* oraz kilka statycznych metod tworzących popularne palety barw. Na podstawie danych z tablicy i palety tworzona jest na żądanie bitmapa w getterze właściwości *Image*. Jeśli brak palety lub zawiera ona więcej niż 256 kolorów, na podstawie danych tworzona jest bitmapa RGB. Operacja rekonstrukcji bitmapy jest bardzo szybka, ponieważ tablica *Pixels* jest kopiowana do danych bitmapy. Właściwość *Image* posiada również setter, który tworzy tablicę *Pixels* na podstawie danych bitmapy.

```
public Bitmap Image
{
    get
    {
        PixelFormat pf = PixelFormat.Format32bppArgb;
        if (Palette != null && Palette.Length <= 256)
            pf = PixelFormat.Format8bppIndexed;
        Bitmap bmp = new Bitmap(Width, Height, pf);
        Rectangle rc = new Rectangle(0, 0, Width, Height);
        BitmapData data = bmp.LockBits(rc, ImageLockMode.WriteOnly, pf);
        byte[] buffer = new byte[data.Stride * Height];
        if (Palette == null)
            Buffer.BlockCopy(Pixels, 0, buffer, 0, Pixels.Length * sizeof(int));
        else
        {
            if (Palette.Length <= 256)
            {
                ColorPalette pal = bmp.Palette;
                Array.Copy(Palette, pal.Entries, Palette.Length);
                bmp.Palette = pal;
                for (int y = 0; y < Height; y++)
                    for (int x = 0; x < Width; x++)
                        buffer[y * data.Stride + x] = (byte)Pixels[y, x];
            }
            else
            {
                for (int y = 0; y < Height; y++)
                    for (int x = 0; x < Width; x++)
                    {
                        Color color = Palette[Pixels[y, x]];
                        int idx = y * data.Stride + 4 * x;
                        buffer[idx + 0] = color.B;
                        buffer[idx + 1] = color.G;
                        buffer[idx + 2] = color.R;
                        buffer[idx + 3] = color.A;
                    }
            }
```

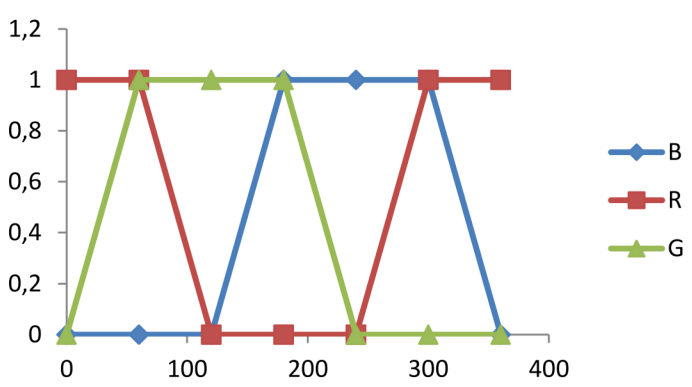
```
            }
            Marshal.Copy(buffer, 0, data.Scan0, buffer.Length);
            bmp.UnlockBits(data);
            return bmp;
        }
        set
        {
            Bitmap bmp = value;
            PixelFormat pf = PixelFormat.Format32bppArgb;
            if (bmp.PixelFormat == PixelFormat.Format8bppIndexed)
                pf = PixelFormat.Format8bppIndexed;
            Rectangle rc = new Rectangle(0, 0, Width, Height);
            BitmapData data = bmp.LockBits(rc, ImageLockMode.ReadOnly, pf);
            byte[] buffer = new byte[data.Stride * Height];
            Marshal.Copy(data.Scan0, buffer, 0, buffer.Length);
            if (bmp.PixelFormat == PixelFormat.Format8bppIndexed)
            {
                ColorPalette pal = bmp.Palette;
                Palette = new Color[pal.Entries.Length];
                for (int i = 0; i < Palette.Length; i++)
                    Palette[i] = pal.Entries[i];
                for (int y = 0; y < Height; y++)
                    for (int x = 0; x < Width; x++)
                        Pixels[y, x] = buffer[y * data.Stride + x];
            }
            else
                Buffer.BlockCopy(buffer, 0, Pixels, 0, buffer.Length);
            bmp.UnlockBits(data);
        }
    }
```

Popularne palety barw zostały zaimplementowane w klasie *Pixelmap* poprzez kombinacje komponentów R, G, B o wartościach zmieniających się zgodnie z liniową funkcja sklejaną z Rys. 27 i przesuniętych względem siebie o parametr *offset*. Jest to funkcja cykliczna z okresem 360°.



Rys. 27. Bazowe funkcje sklejane komponentów R, G, B w paletach barw

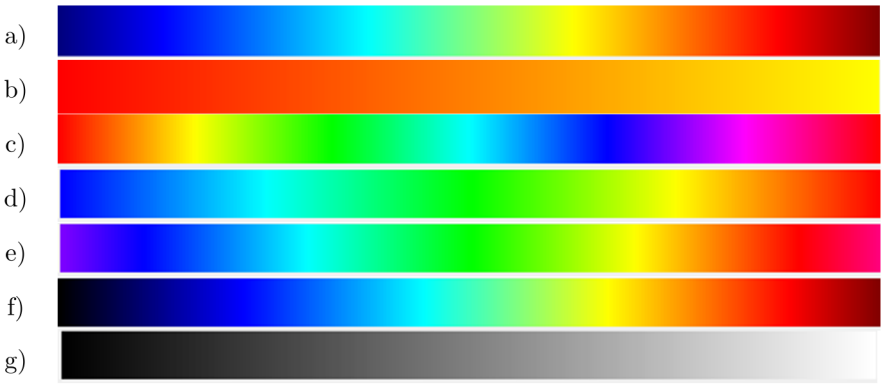
Ze względu na okresowość funkcji, przebieg reprezentujący wartości komponentów RGB w przedziale (hue, hue + 360) wygląda tak, jak na Rys. 28.



Rys. 28. Funkcje sklejane komponentów R, G, B z uwzględnieniem okresowości

Dla *offset* = 0° uzyskuje się paletę szarości, dla 60° popularne palety typu Jet, a dla 120° palety typu Spectrum (Rys. 29). Parametr *hue* decyduje o zakresie odcieni barw.

```
public static Color[] CreatePalette(int startHue, int stopHue, int offset = 120)
{
    Color[] pal = new Color[256];
    for (int i = 0; i < pal.Length; i++)
    {
        double hue = startHue + (stopHue - startHue) * i / 255.0;
        int r = Linearize(hue + offset, offset);
        int g = Linearize(hue, offset);
        int b = Linearize(hue - offset, offset);
        pal[i] = Color.FromArgb(r, g, b);
    }
    return pal;
}

public static int Linearize(double hue, double offset)
{
    double result = 0;
    if (hue < 0) hue += 360;
    if (hue > 360) hue -= 360;
    double[] tresh = new double[] { 0, 60, 60 + offset, 120 + offset };
    if (hue < tresh[1])
        result = (hue - tresh[0]) / (tresh[1] - tresh[0]);
    else if (hue < tresh[2])
        result = 1;
    else if (hue < tresh[3])
        result = (tresh[3] - hue) / (tresh[3] - tresh[2]);
    return (int)(255 * result);
}

public static Color[] Jet { get { return CreatePalette(210, -30, 60); } }
public static Color[] JetHot { get { return CreatePalette(240, -30, 60); } }
public static Color[] JetHSV { get { return CreatePalette(240, 0); } }
public static Color[] Autumn { get { return CreatePalette(0, 60); } }
public static Color[] HSV { get { return CreatePalette(0, 360); } }
public static Color[] Cool { get { return CreatePalette(270, 180); } }
public static Color[] Summer { get { return CreatePalette(60, 120); } }
public static Color[] Rainbow { get { return CreatePalette(270, -30); } }
public static Color[] Grey { get { return CreatePalette(0, 60, 0); } }
```

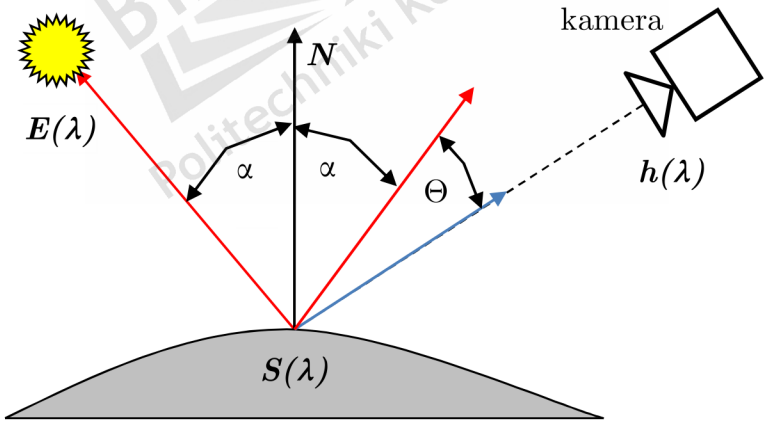


Rys. 29. Przykładowe palety barw uzyskane za pomocą klasy Pixelmap: a) Jet b) Autumn c) HSV d) JetHSV e) Rainbow f) JetHot g) Grey

Ze względu na rozszerzoną funkcjonalność, klasa *Pixelmap* zastąpiła w projekcie klasę Bitmap i użyta została m.in do zapamiętania obrazu przed i indeksowanego obrazu po segmentacji, obrazów segmentów przetwarzanych przez sieć SSN, itp.

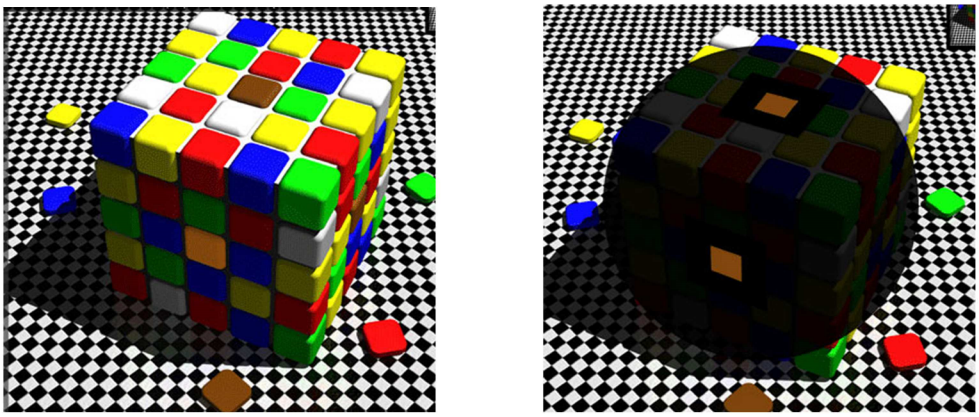
3.3 Balans bieli

Fala świetlna padająca na powierzchnię obiektu jest częściowo absorbowana. Poziom absorpcji jest funkcją częstotliwości. Część promieniowania ulega odbiciu i jest odpowiedzialna za kontrast optyczny (Rys. 30) [28].



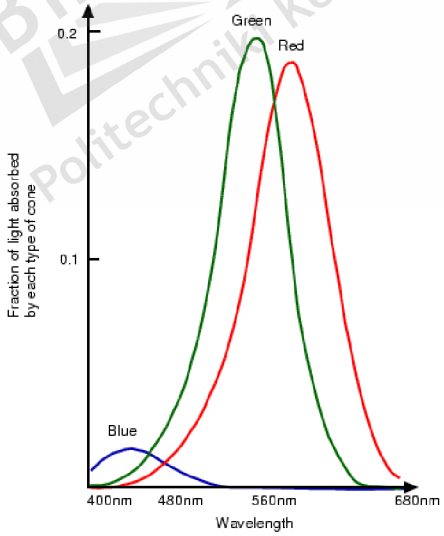
Rys. 30. Podstawowy model oświetlenia

Zgodnie z teorią Retinex, którą przedstawił w latach 1950-tych Edwin Land, człowiek potrafi rozpoznawać kolory przy różnych warunkach oświetlenia. Jest to zjawisko określane jako "color constancy". W „widzeniu komputerowym" staramy się również uniezależnić kolor obiektów od koloru oświetlenia przy użyciu automatycznych algorytmów balansu bieli. Teoria Retinex udowadnia, że wrażenie wzrokowe w dużym stopniu zależy także od kontekstu. Słynne są zwłaszcza przykłady z laboratorium Beau Lotto (Rys. 31).



Rys. 31. Zależność rozpoznawania kolorów od kontekstu. Przykład z laboratorium Beau Lotto

Widmo światła docierającego do obiektywu kamery jest rozdzielane za pomocą filtrów na trzy podzakresy, określające w przybliżeniu intensywność światła wokół częstotliwości fali odpowiadającej czerwieni R (700 nm), zieleni G (545 nm) i błękitowi B (480 nm). Detektory w kamerach i aparatach fotograficznych rejestrują przeważnie intensywność światła z użyciem filtrów Bayera w celu określenia udziału komponentów RGB w świetle. Intensywność każdej składowej jest proporcjonalna do zintegrowanej odpowiedzi sensora w zakresie widma widzialnego. Taka analiza trójchromatyczna jest wystarczająca do odtworzenia wrażenia koloru (tristimulus theory [29], Rys. 32).



Rys. 32. Wrażliwości sensorów oka ludzkiego (czopków) na składowe R, G, B

Intensywności składowych rejestrowanego koloru można wyznaczyć jako:

$$[R, G, B] \propto \int S(\lambda)E(\lambda)h_{R,G,B}(\lambda)d\lambda \tag{3.23}$$

gdzie:

$E(\lambda)$ jest dystrybucją widma mocy promieniowania,
$S(\lambda)$ – współczynnikiem odbicia oświetlanej powierzchni
$h_{R,G,B}(\lambda)$ kolejnymi transmitancjami (wrażliwościami) filtrów sensorów R, G i B, zwanymi również funkcjami dopasowania koloru.

Ze względu na liniowość równania (3.23), zgodnie z prawem Grassmana, w wyniku zmieszania źródeł światła o kolorach określonych współrzędnymi R1, G1, B1 oraz R2, G2, B2, postrzegany kolor ma współrzędne RGB będące sumą poszczególnych składowych:

$$[R, G, B] = [R1 + R2, G1 + G2, B1 + B2] \tag{3.24}$$

Zakładając niezależność dystrybucji widma mocy źródła światła i współczynnika odbicia, można przyjąć, że:

$$[R, G, B] = \int S(\lambda) h_{R,G,B}(\lambda) d\lambda \int E(\lambda) h_{R,G,B}(\lambda) d\lambda = [R^S R^E, G^S G^E, B^S B^E]$$

Rzeczywisty kolor obiektu R^S, G^S, B^S zależy m.in. od widma oświetlenia R^E, G^E, B^E. Dlatego dla światła, które nie jest idealnie białe, wymagana jest korekcja, polegająca na znalezieniu współczynników skalujących. Jest ona związana ze znalezieniem wektora odchylenia koloru białych obiektów na obrazie od linii szarości w modelu RGB i nazywana jest balansem bieli. Kamery posiadają wbudowane systemy automatycznego balansu bieli, który w zależności od użytego algorytmu i panujących warunków lepiej lub gorzej spełnia swoje zadanie. Stosunkowo łatwo jest przeprowadzić korekcję posługując się tablicami referencyjnymi kolorów (Macbeth color checker, IT8, white target). Ponieważ jednak każdorazowa kalibracja jest uciążliwa, wagi korekcji zapamiętywane są dla typowych warunków oświetlenia (światło dzienne, tungsten, fluorescencyjne). Możliwe jest również dokonanie automatycznego pomiaru balansu bieli na podstawie obrazu lub sekwencji wideo z wykorzystaniem skalowania składowych koloru.

$$[R^S, G^S, B^S] = \left[\frac{R}{R^E}, \frac{G}{G^E}, \frac{B}{B^E}\right] \tag{3.25}$$

Dla światła zbliżonego do bieli, gdy $R^E \approx G^E \approx B^E$, jasność obrazu po korekcji

$$I^S = R^S + G^S + B^S = \frac{R}{R^E} + \frac{G}{G^E} + \frac{B}{B^E} \approx \frac{R + G + B}{\frac{1}{3}(R^E + G^E + B^E)} = \frac{I}{\frac{1}{3} I^E}$$

Balansu bieli można zatem dokonać zachowując jego jasność, jeśli

$$\frac{1}{3}(R^E + G^E + B^E) = 1 \tag{3.26}$$

Aby uniezależnić się od liczby analizowanych pikseli, można określić składowe światła na podstawie unormowanych wag:

$$[R^E, G^E, B^E] = \frac{w_{R,G,B}}{\frac{1}{3}\sum_{i=R,G,B} w_i} \tag{3.27}$$

Dla popularnych metod automatycznego balansu bieli [30] [31] wagi te można wyznaczyć wg sposobów opisanych poniżej.

3.3.1 GWT (Grey World Theory)

Najstarsza i najprostsza metoda zakłada, że statystycznie średni kolor piksela obrazu oświetlonego białym światłem jest szary. Średni kolor przybliża zatem kolor źródła światła. Ze względu na normalizację wag, można je wyznaczyć jako sumę wartości wszystkich pikseli w obrazie, oddzielnie dla każdej warstwy R, G i B.

$$w_{R,G,B} = \sum R, G, B \tag{3.28}$$

3.3.2 White Patch (RGB Max)

Najjaśniejsze piksele zawierają najdokładniejszą informację o kolorze źródła światła. Stanowią one koniec wektora odchylenia od linii szarości i wprowadzają najmniejszy błąd do obliczeń. W szczególności odbicia zwierciadlane nie są związane z kolorem obiektu. Wagi mogą być określone jako suma z pewnego procentu najjaśniejszych pikseli w obrazie:

$$w_{R,G,B} = \sum max(R), max(G), max(B) \tag{3.29}$$

3.3.3 Shades of Grey

W zależności od warunków oświetlenia lepsza okazuje się metoda GWT lub RGB Max, dlatego powstała technika pośrednia, korzystająca z tego, że oba równania (3.28) i (3.29) można zapisać przy użyciu normy Minkowskiego. Dla p = 1 otrzymujemy wówczas wagi GWT, a dla p = ∞, wagi RGB Max.

$$w_{R,G,B} = \sqrt[p]{\sum R^p, G^p, B^p} \tag{3.30}$$

Najpopularniejszymi wyborami parametru p okazuje się zakres od 4 do 6. Przykład prostej implementacji:

```
double[] ShadesOfGrey(int p)
{
    double[] bias = new double[3];
    for (int i = 0; i < Segmenter.Root.Children.Count; i++)
```

```
        {
            TSegment seg = Segmenter.Root.Children[i];
            bias[0] += Math.Pow(seg.Color.R, p);
            bias[1] += Math.Pow(seg.Color.G, p);
            bias[2] += Math.Pow(seg.Color.B, p);
        }
        bias[0] = Math.Pow(bias[0], 1.0 / p);
        bias[1] = Math.Pow(bias[1], 1.0 / p);
        bias[2] = Math.Pow(bias[2], 1.0 / p);
        return bias;
    }
```

3.3.4 Grey Edge

Metoda wykorzystuje analizę koloru krawędzi w obrazie. Hipoteza zakłada, że średni kolor krawędzi jest związany z kolorem oświetlenia. Do wyznaczenia wag wykorzystywane są piksele po filtracji górnoprzepustowej obrazu. Zakładając, że dla *i*-tego piksela *dist[i]* oznacza sumę odległości koloru tego piksela od kolorów pikseli sąsiednich, wagi można wyznaczyć jako sumę wartości pikseli ważonych odległościami *dist[i]*.

$$w_{R,G,B} = \sum_i dist[i] \cdot [\ R[i], G[i], B[i]\ ] \tag{3.31}$$

3.3.5 Grey Edge & Shades of Grey

W Systemie dostępna jest opcja automatycznej kalibracji balansu bieli na podstawie uśrednienia wyników wag z kilkunastu ostatnich obrazów. Wykorzystany algorytm jest połączeniem Grey Edge i Shades of Grey. Wartość każdego piksela jest bowiem skalowana sumą odległości od jego sąsiadów (GE) i potęgowana tak, jak w algorytmie Shades of Grey. W ten sposób, piksele jasne oraz znajdujące się na krawędziach otrzymują wyższe wagi (a najwyższą uzyskują odbłyski na krawędziach). Kalibracja balansu pozwala również na dokładniejszą segmentację obrazu statycznego. Wystarczy zatrzymać kamerę na wybranym kadrze, a zakładając, że scena jest statyczna, balans bieli uwypukli kontrast pomiędzy pikselami a wartością średnią koloru w kadrze.

$$w_{R,G,B} = \sqrt[p]{\sum_i (dist[i] \cdot [\ R[i], G[i], B[i]\ ])^p} \tag{3.32}$$

```
double[] GreyEdge(int p)
{
    double[] bias = new double[3];
    for (int i = 0; i < Segmenter.Root.Children.Count; i++)
    {
        TSegment seg = Segmenter.Root.Children[i];
        double dist = 0;
        for (int j = 0; j < seg.Neighbors.Count; j++)
            dist += seg.DistanceTo(seg.Neighbors[j]);
        bias[0] += Math.Pow(dist * seg.Color.R, p);
        bias[1] += Math.Pow(dist * seg.Color.G, p);
```

```
                bias[2] += Math.Pow(dist * seg.Color.B, p);
            }
            bias[0] = Math.Pow(bias[0], 1.0 / p);
            bias[1] = Math.Pow(bias[1], 1.0 / p);
            bias[2] = Math.Pow(bias[2], 1.0 / p);
            return bias;
        }
```

3.4 Filtracja statystyczna

Segmentacja obrazu jest oparta przede wszystkim na porównywaniu składowej odcienia kolorów pikseli. Piksele o słabym nasyceniu określają jednak odcień z dużym błędem, wprowadzając szum do warstwy hue obrazu. O odcieniu obszarów słabo nasyconych można wnioskować na podstawie analizy statystycznej. Filtracja statystyczna przeprowadzona na warstwie hue może poprawić segmentację, choć warto zauważyć, że segmentacja wododziałowa jest również metodą statystyczną (analizuje otoczenie piksela i wybiera piksel najbardziej podobny). Część wododziałowa segmentacji może podczas porównywania segmentów brać pod uwagę zamiast kolorów segmentów, histogramy otoczenia porównywanych segmentów. Prowadzi to do segmentacji opartej na analizie tekstury.

3.5 Podsumowanie

Rozdział przedstawia modyfikacje i implementacje znanych algorytmów interpolacji biliniowej i bikubicznej, które są wykorzystane przy zmianie rozdzielczości obrazów wejściowych, definicji wag funkcji porównującej kolory, w celu uzyskania quasi-ciągłej reprezentacji funkcji HRTF, itd. i mają charakter uniwersalny. Interpolacja liniowa składowych RGB wykorzystana została do implementacji klasy *Pixelmap* umożliwiającej posługiwanie się obrazami indeksowanymi z paletami pseudokolorów, oraz wydajne przetwarzanie obrazów indeksowanych i RGB w środowisku .NET. Przetwarzanie wstępne obrazów, oprócz przeskalowania do standardowej rozdzielczości, obejmuje automatyczny balans bieli. Przedstawione zostały popularne algorytmy GWT, RGB Max, Shades of Grey i Grey Edge, oraz połączenie dwu ostatnich jako rozwiązanie własne.

4. Segmentacja obrazu

Celem segmentacji obrazu jest grupowanie pikseli ze zbioru MxN-elementowego na zbiór o mniejszej liczbie elementów k, minimalizująca pewną funkcję kosztu będącą błędem segmentacji. Z tego punktu widzenia segmentacja jest przekształceniem podobnym do aproksymacji, ale operującym na zbiorach. Często nie zakłada się z góry liczności k zbioru docelowego, ale uzależnia ją od żądanego stopnia uszczegółowienia i rozkładu wartości danych, dlatego warto zaimplementować możliwość rekurencyjnej segmentacji obrazów już posegmentowanych do obrazów z mniejszą liczbą detali. Wynikiem takiego przetwarzania jest struktura hierarchiczna. Zasadniczą segmentację może poprzedzać analiza czynników głównych PCA, która umożliwia ortogonalizację przestrzeni wartości pikseli segmentowanego obrazu. Metody segmentacji najczęściej dzieli się na metody pikselowe, które podczas łączenia pikseli biorą pod uwagę wyłącznie wartość przetwarzanego piksela i metody kontekstowe, które uwzględniają również wartości pikseli sąsiednich – z otoczenia przetwarzanego piksela. Segmentacja jest problemem NP-trudnym nawet w przypadku podziału pojedynczego obrazu na dwa segmenty.

4.1 Implementacja algorytmów segmentacji

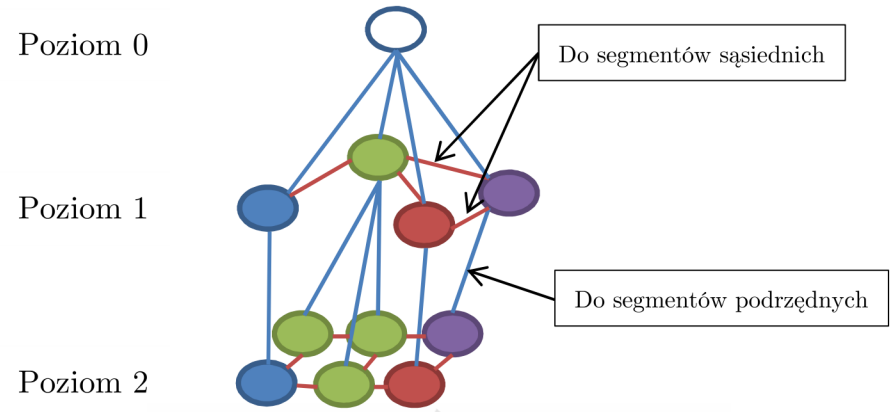
Dostępne metody segmentacji obrazu zaimplementowane zostały w Systemie jako metody klasy *TSegmenter*. Domyślnym algorytmem jest hybrydowy algorytm wododziałowo-rozrostowy, jednak istnieje możliwość wyboru m.in spośród algorytmów: k-means, rozrostu, wododziałowego i split & merge. W celu zachowania po segmentacji informacji o sąsiedztwie segmentów, wykorzystana została struktura drzewa, którego elementami są obiekty klasy *TSegment* zawierające listę podsegmentów *Children* oraz sąsiadów *Neighbors*. Z obiektem segmentera związane są również dwa obiekty klasy *Pixelmap*. Pixelmapa źródłowa *Src* podczas przypisania w setterze tworzy najwyższy poziom w drzewie hierarchii segmentów wstawiając nowe segmenty utworzone na podstawie wartości i położenia pikseli. Jednocześnie do listy *Neighbors* dołączane są segmenty sąsiednie. Pikselmapa docelowa *Dest* tworzy na żądanie bitmapy indeksowane z wynikiem segmentacji z najwyższego poziomu hierarchii segmentów. Ponieważ w wyniku segmentacji obiekty często zmieniają rodzica, operacje in-place na tej samej liście-tablicy dzieci rodzica nie dają się efektywnie implementować (poza implementacją listy wiązanej, która nie zapewnia jednak dostępu swobodnego do elementów takiej listy). Szczególnie kosztowna jest operacja usunięcia segmentu z listy dzieci jego rodzica (trzeba dany segment w niej odnaleźć, usunąć i przesunąć wszystkie segmenty następne w tej liście). Obliczenia można jednak znacznie przyspieszyć tworząc podczas segmentacji nowy korzeń

hierarchii i nowych rodziców (brak operacji usuwania pojedynczych segmentów z listy). W ten właśnie sposób zostały zaimplementowane dostępne algorytmy segmentacji w Systemie.

```
Pixelmap FDest;
public Pixelmap Dest
{
    get
    {
        if (FDest == null)
        {
            FDest = new Pixelmap(FSrc.Height, FSrc.Width);
            Root.Draw(FDest);
        }
        return FDest;
    }
    set { FDest = value; }
}
Pixelmap FSrc;
TSegment[,] Segments;
public Pixelmap Src
{
    get { return FSrc; }
    set
    {
        FSrc = value;
        Width = value.Width;
        Height = value.Height;
        Root.Children.Clear();
        Segments = new TSegment[Width, Height];
        for (int w = 0; w < Width; w++)
            for (int h = 0; h < Height; h++)
            {
                TSegment child = new TSegment();
                child.Origin = new Point(w, h);
                child.Color = value[h, w];
                child.Parent = Root;
                Segments[w, h] = child;
            }
        int U = 3;
        int V = 3;
        for (int w = 0; w < Width; w++)
            for (int h = 0; h < Height; h++)
                for (int u = 0; u < U; u++)
                    for (int v = 0; v < V; v++)
                    {
                        int x = w + u - U / 2;
                        int y = h + v - V / 2;
                        if (x >= 0 && x < Width && y >= 0 && y < Height)
                            Segments[w, h].Neighbors.Add(Segments[x, y]);
                    }
    }
}
```

Każda metoda segmentacji grupuje segmenty z najwyższego poziomu drzewa hierarchii tworząc nowy poziom powyżej, o mniejszej liczbie segmentów. Lista *Neighbors* nowych segmentów jest uzupełniana podczas segmentacji, tak, by segmenty nadal posiadały informację o segmentach sąsiednich. Może być ona wykorzystana do redukcji mniejszych segmentów przez łączenie ich z sąsiednim, najbardziej do nich

podobnym (patrz Redukcja). Jest to również rodzaj segmentacji, w której tworzona jest nowa, najwyższa warstwa w drzewie hierarchii. Dzięki strukturze drzewa, wynik segmentacji z większym poziomem szczegółów, może być danymi źródłowymi dla kolejnego algorytmu segmentacji, pomniejszającym poziom szczegółów. Możliwy jest również teoretycznie dostęp do każdej z warstw uszczegółowienia (Rys. 33).



Rys. 33. Struktura połączeń pomiędzy segmentami w drzewie

Podczas przypisywania nowemu segmentowi (obiekt typu *TSegment*) jego rodzica do property *Parent*, następuje automatyczne dopisanie segmentu do listy *Children* przyszłego rodzica oraz zsumowanie składowych koloru agregacyjnego rodzica i segmentu. Jednocześnie sumowana jest liczba pikseli dodawanych do koloru agregacyjnego, by w każdej chwili móc efektywnie wyznaczyć średni kolor segmentu. Średni kolor segmentu służy jako kolor referencyjny w algorytmie rozrostu obszaru, ale także jako etykieta segmentu dla prezentacji segmentacji bez użycia palet pseudokolorów.

4.2 Wektory i wartości własne

Dla dowolnej macierzy kwadratowej A o rozmiarze n, wektor y nazywamy wektorem własnym (eigenvector), a liczbę λ wartością własną (eigenvalue) tej macierzy, jeśli:

$$\boldsymbol{A}y = \lambda y \tag{4.1}$$

Dla wektora własnego y macierz transformacji liniowej A nie zmienia jego kierunku (np. wektor osi obrotu nie zmienia kierunku podczas obrotu). Z równania (4.1) wynika, że macierz $\boldsymbol{A} - \lambda \boldsymbol{I}$ jest nieodwracalna (singular matrix), a wektory własne leżą w jądrze (null space) tej macierzy.

$$(\boldsymbol{A} - \lambda \boldsymbol{I})y = 0$$

Wiedząc, że wyznacznik tej macierzy jest równy zeru, można wyznaczyć wartości własne macierzy A, a na ich podstawie również wektory własne. Określenie ich na tej

podstawie wymaga znalezienia pierwiastków wielomianu charakterystycznego n-tego stopnia. Dla n $>$ 4, zgodnie z teorią Abela-Ruffiniego, nie jest możliwe algebraiczne określenie pierwiastków wielomianu dla dowolnych jego współczynników. Do ich wyznaczenia konieczne jest użycie iteracyjnych metod numerycznych, np. Newtona-Raphsona, Jenkins-Traub, Laguerra. W praktyce jednak do wyznaczenia wartości i wektorów własnych wykorzystuje się dużo efektywniejsze i dokładniejsze metody numeryczne - iteracji potęg lub faktoryzacji QR macierzy. Ze względu na złe uwarunkowania, to właśnie problem znalezienia pierwiastków wielomianu dowolnego stopnia przedstawiany jest częściej jako zadanie wyznaczenia wartości własnych macierzy towarzyszącej [32].

Macierz A posiada n wartości (i wektorów) własnych. Suma wartości własnych macierzy jest równa śladowi macierzy (sumie elementów na przekątnej), natomiast ich produkt (podobnie jak iloczyn elementów głównych macierzy w dekompozycji LU) jest równy wyznacznikowi macierzy.

$$\sum_{i=1}^{n} \lambda_i = \sum_{i=1}^{n} A_{ii} = trace(A) \tag{4.2}$$

$$\prod_{i=1}^{n} \lambda_i = det(A) \tag{4.3}$$

Dodatnich (lub ujemnych) wartości własnych jest tyle samo, co dodatnich (lub ujemnych) elementów głównych macierzy.

4.2.1 Diagonalizacja

Zestawiając wektory własne w kolumny macierzy **S** i zakładając, że wszystkie są liniowo niezależne (istnieje macierz odwrotna $\boldsymbol{S}^{-1}$**),** można dokonać faktoryzacji macierzy **A** zwanej diagonalizacją, ze względu na diagonalną macierz wartości własnych $\boldsymbol{\Lambda}$

$$\boldsymbol{AS} = \boldsymbol{A}[y_1 \quad \cdots \quad y_n] = [\lambda_1 y_1 \quad \ldots \quad \lambda_n y_n] = \boldsymbol{S} \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \ldots & 0 \\ 0 & 0 & \lambda_n \end{bmatrix} = \mathbf{S\Lambda} \tag{4.4}$$

$$\boldsymbol{A} = \boldsymbol{S\Lambda S}^{-1} \tag{4.5}$$

Diagonalizacja określa zmianę układu współrzędnych do bazy określonej przez wersory wektorów własnych. Zakładając, że $\boldsymbol{A}x = b$ i $\boldsymbol{A} = \boldsymbol{S\Lambda S}^{-1}$:

$$\boldsymbol{\Lambda S}^{-1}x = \boldsymbol{S}^{-1}b \tag{4.6}$$

Wektor x i wynikowy wektor b można zinterpretować jako liniową kombinację kolumn macierzy S, czyli wektorów własnych.

Ze względu na istnienie pewnej odwracalnej operacji liniowej **M**, macierz:

$$\boldsymbol{B} = \boldsymbol{M^{-1}AM} \tag{4.7}$$

jest macierzą podobną do **A**, co oznacza, że **B** ma takie same wartości własne jak macierz **A**. Dowód:

$$\boldsymbol{M^{-1}}\boldsymbol{A}y = \boldsymbol{M^{-1}}\lambda y$$

$$(\boldsymbol{M^{-1}AM})\boldsymbol{M^{-1}}y = \lambda \boldsymbol{M^{-1}}y$$

$$\boldsymbol{B}(\boldsymbol{M^{-1}}y) = \lambda(\boldsymbol{M^{-1}}y) \blacksquare$$

Wektorami własnymi macierzy B są przetworzone operacją liniową $\boldsymbol{M^{-1}}$ wektory własne macierzy A.

4.2.2 Iteracja potęg

Potęgowanie macierzy A nie zmienia wektorów własnych, zaś wartości własne są potęgami wartości własnych oryginalnej macierzy:

$$\boldsymbol{A}^k y = \lambda^k y \tag{4.8}$$

$$\boldsymbol{A}^k = \boldsymbol{S\Lambda}^k \boldsymbol{S}^{-1} \tag{4.9}$$

W szczególności:

$$\boldsymbol{A}^2 = \boldsymbol{S\Lambda S}^{-1}\boldsymbol{S\Lambda S}^{-1} = \boldsymbol{S\Lambda}^2\boldsymbol{S}^{-1}$$

$$\boldsymbol{A}^{-1} = \boldsymbol{S\Lambda}^{-1}\boldsymbol{S}^{-1}$$

Wektory własne macierzy odwrotnej są takie same jak oryginalnej macierzy, zaś wartości własne są odwrotnością wartości własnych oryginalnej macierzy.

Ponieważ wyznaczenie wartości własnych na podstawie pierwiastków wielomianu charakterystycznego stopnia wyższego niż czwarty wiąże się z koniecznością wykorzystania metod numerycznych, efektywniejszą i dokładniejszą metodą jest wyznaczenie wektorów własnych za pomocą tzw. iteracji potęg (power iteration). Dla dowolnego wektora x o elementach niezerowych, ciąg wektorów

$$Ax, A^2x, \ldots, A^kx$$

dąży do wektora własnego związanego z największą wartością własną macierzy A. Aby potwierdzić tę obserwację należy zauważyć, że wektor x można określić jako kombinację liniową jednostkowych wektorów własnych:

$$x = \sum_{i=1}^{n} c_i y_i$$

Wówczas, dla $k \to \infty$ i posortowanych wartości własnych $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$:

$$A^k x = \sum_{i=1}^{n} c_i A^k y_i = \sum_{i=1}^{n} c_i \lambda_i^k y_i \approx c_1 \lambda_1^k y_1$$

Wynikowy wektor jest zatem wyskalowanym wektorem własnym, związanym z największą wartością własną. Ze względu na możliwość przekroczenia zakresu wartości podczas potęgowania, w każdej iteracji wektor własny jest poddawany normalizacji. Nie musi być to norma euklidesowa, przeważnie jest to prostsza obliczeniowo norma Manhattan lub max.

$$y_1 \approx \frac{A^k x}{\|A^k x\|} \tag{4.10}$$

Szybkość zbieżności algorytmu zależy liniowo od stosunku drugiej do pierwszej dominującej wartości własnej $|\lambda_2|/|\lambda_1|$. Jest to bardzo popularny algorytm do wyznaczania dominujących wartości własnych dla bardzo dużych, ale rzadkich macierzy, np. firma Google używa go do pozycjonowania stron w swojej wyszukiwarce internetowej, Twitter – do sugerowania śledzenia linków do innych użytkowników, itd.

```
function [lambda,x] = PowerIter(A,x)
err = inf;
while err > eps
    xPrev = x;
    x = x / norm(x,inf);
    x = A * x;
    err = norm(xPrev - x);
end
x = x / norm(x);
lambda = x' * A * x;
```

Jeśli zamiast macierzy A będziemy badać zbieżność ciągu dla A^{-1}, okaże się, że metoda wyznacza wektor własny związany z największą wartością własną A^{-1}, której odwrotność jest jednocześnie najmniejszą eigenwartością A. Zauważmy, że wektory własne macierzy $(A - \mu I)^{-1}$ są takie same jak macierzy A, natomiast wartości własne są odwrotnościami przesuniętych wartości własnych macierzy A:

$$Ax = \lambda x + \mu x - \mu x$$

$$(A - \mu I)x = (\lambda - \mu)x$$

$$(A - \mu I)^{-1} x = (\lambda - \mu)^{-1} x \ \blacksquare$$

Dla wartości μ bliskich jednej z wartości własnych λ macierzy A, macierz $(A - \mu I)^{-1}$ będzie miała największą wartość własną równą $(\lambda - \mu)^{-1}$, którą można wyznaczyć przy użyciu algorytmu iteracji potęg. Dodatkowo, różnica względem kolejnej wartości własnej będzie również większa od różnicy λ względem sąsiednich wartości własnych macierzy A, a więc algorytm iteracji potęg będzie szybciej zbieżny. W ten sposób, na podstawie oszacowania przesunięcia μ można wyznaczyć dowolną wartość własną macierzy A (inverse power iteration algoritm).

Oszacowaniem wartości λ dla dowolnego wektora x w sensie najmniejszej normy $\|Ax - \lambda x\|$ jest iloraz Rayleigh'a:

$$R(A,x) = \frac{x^H Ax}{x^H x} = \frac{x^H Ax}{\|x\|^2} \tag{4.11}$$

Dla wektora własnego y jest on równy jego wartości własnej:

$$R(A,y) = \frac{y^H Ay}{y^H y} = \frac{y^H \lambda y}{y^H y} = \lambda$$

Wykorzystując iloraz Rayleigha do oszacowania przesunięcia μ w odwrotnej iteracji potęg, otrzymujemy szybką zbieżność, rzędu $O(n^3)$ w algorytmie Rayleigh Quotient Iteration (RQI).

```
function [lambda,x] = RQI(A,x)
n = min(size(A));
x = x / norm(x);
lambda = x' * A * x;
err = inf;
while err > eps
    xPrev = x;
    x = (A - lambda * eye(n)) \ x;
    x = x / norm(x);
    lambda = x' * A * x;
    err = norm(xPrev - x);
end
```


4.2.3 Diagonalizacja macierzy symetrycznych

Dla rzeczywistych macierzy symetrycznych K ($K = K^T$), wektory własne są ortogonalne względem siebie. Ich iloczyn skalarny jest zatem równy zeru. Macierzą odwrotną do macierzy ortogonalnych wektorów własnych Q jest wówczas macierz transponowana Q^T

$$\boldsymbol{K} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T \tag{4.12}$$

Dekompozycja LU (Cholesky) i eliminacja Gaussa dokonuje rozkładu:

$$\boldsymbol{K} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T, \tag{4.13}$$

gdzie L jest macierzą trójkątną, a D – diagonalną.

Jeśli K jest rzeczywistą macierzą symetryczną, jej wartości własne są rzeczywiste. Zapisując równanie (4.1) dla wartości sprzężonych:

$$\overline{\boldsymbol{K}}\bar{y} = \bar{\lambda}\bar{y},$$

a następnie wykonując transpozycję dla obu stron równania i mnożąc przez wektor y:

$$\bar{y}^T \overline{\boldsymbol{K}}^{\boldsymbol{T}} = \bar{y}^T \boldsymbol{K} = \bar{y}^T \bar{\lambda}$$

$$\bar{y}^T \boldsymbol{K} y = \bar{y}^T \bar{\lambda} y$$

$$\bar{y}^T \lambda y = \bar{y}^T \bar{\lambda} y$$

$$\lambda = \overline{\lambda} \blacksquare$$

Wartości własne macierzy K są takie same jak ich sprzężenia – a więc są rzeczywiste.

4.2.4 Kombinacja liniowa projekcji

Diagonalizację macierzy symetrycznej można również przedstawić jako kombinację liniową macierzy projekcji (rzędu 1):

$$\boldsymbol{K} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T = [q_1 \quad \cdots \quad q_n] \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & \lambda_n \end{bmatrix} \begin{bmatrix} {q_1}^T \\ \dots \\ {q_n}^T \end{bmatrix} = \lambda_1 q_1 {q_1}^T + \cdots + \lambda_n q_n {q_n}^T$$

Po wyznaczeniu dominującej wartości i wektora własnego macierzy K za pomocą iteracji potęg, można zmodyfikować macierz K odejmując czynnik $\lambda_1 q_1 {q_1}^T$ i powtórzyć algorytm wyznaczając kolejną parę wartości i wektorów własnych. Postępując w ten sposób można teoretycznie wyznaczyć wszystkie pary wartości i wektorów własnych. Błąd numeryczny wyznaczania kolejnych par kumuluje się, dlatego konieczna jest ortogonalizacja wyznaczanych wektorów własnych (np. za pomocą algorytmu Gram-Schmidta).

4.2.5 Algorytm QR

Istnieje jednak prostszy i dokładniejszy algorytm, wykorzystujący ideę iteracji potęg, który pozwala dokonać diagonalizacji macierzy przy użyciu faktoryzacji QR [33] [34] – algorytm (iteracja) QR [35]:

$$\begin{aligned} Q_k R_k &= A_k \\ A_{k+1} &= R_k Q_k \end{aligned} \tag{4.14}$$

W każdej iteracji wyznaczana jest nowa macierz A_{k+1}, która jest podobna do A:

$$A_{k+1} = R_k Q_k = Q_k^T A_k Q_k = (Q_1 \dots Q_k)^T A_1 Q_1 \dots Q_k$$

Dla $k \to \infty$,

$$A_\infty = Q_\infty^T A_\infty Q_\infty$$

A więc, jeśli ciąg jest zbieżny:

$$Q_\infty = I, A_\infty = R_\infty$$

Macierz A_{k+1} dąży zatem do postaci macierzy trójkątnej, a w przypadku symetrycznej macierzy A – postaci diagonalnej, zachowując te same wartości własne jak startowa macierz A. Macierz wektorów własnych można odtworzyć z produktu

macierzy ortogonalnych kolejnych iteracji, bowiem stanowi ona ortogonalną bazę kolejnych potęg macierzy A:

$$A^k = Q_1 \dots Q_k R_k \dots R_1$$

```
function [S] = QRiter(A)
err = inf;
while err > eps * norm(A)
    Aprev = A;
    [Q,R] = qr(A);
    A = R*Q;
    err =  norm(A - Aprev, 'fro');
end
S = A;
```

Algorytm w tej postaci, podobnie jak iteracja potęg, jest wolno zbieżny. W celu przyspieszenia zbieżności, analogicznie do algorytmu RQI, w każdej iteracji stosuje się przesunięcie μ przekątnych macierzy A_k będące oszacowaniem wartości własnej na podstawie ilorazu Rayleigha lub stabilniejszym, tzw. przesunięciem Wilkinsona [36]. Iloraz Rayleigha można wprost odczytać z przekątnej macierzy A_k:

$$\begin{aligned} \mu &= A_k[n,n] \\ Q_k R_k &= A_k - \mu I \\ A_{k+1} &= R_k Q_k + \mu I \end{aligned} \tag{4.15}$$

W każdej iteracji wyznaczana jest nowa macierz A_{k+1}, która jest podobna do A:

$$A_{k+1} = R_k Q_k + \mu I = Q_k^T (A_k - \mu I) Q_k + \mu I = Q_k^T A_k Q_k$$

Po ustaleniu wartości własnej μ wykonywana jest deflacja macierzy, czyli redukcja jej wymiaru i wyznaczana jest kolejna wartość własna.

4.3 Analiza czynników głównych PCA

Metoda czynników głównych PCA dokonuje transformacji liniowej sygnału wielowymiarowego, określonego na podstawie skorelowanych ze sobą wersorów parametrów, do nowego układu współrzędnych, o nieskorelowanych parametrach i maksymalnej wariancji sygnału w kierunku wersorów [37] [38] [39]. Brak korelacji wersorów parametrów oznacza ich ortogonalność. PCA pozwala znaleźć istotne z punktu widzenia opisu cech obiektu nowe parametry, będące liniową kombinacją pierwotnych parametrów. Metoda stara się zatem odsłonić prawdziwą strukturę danych poprzez analizę ich wariancji. PCA umożliwia m.in redukcję wymiarowości danych poprzez odrzucenie wersorów o małym wpływie na wartości sygnału (małych wariancjach) – dane rzutowane są wówczas ortogonalnie na pozostałe wymiary. Ułatwia również segmentację danych dzięki rozkładowi na wersory niezależne (ortogonalne).

Dla macierzy $\mathbf{X}$ o rozmiarze $m \times n$, której wiersze stanowią wektory obserwacji, a w kolumnach zapisane są wartości parametrów opisujących te obserwacje, PCA dokonuje wycentrowania wartości parametrów w każdej kolumnie, w celu łatwiejszego wyznaczenia wariancji obserwacji.

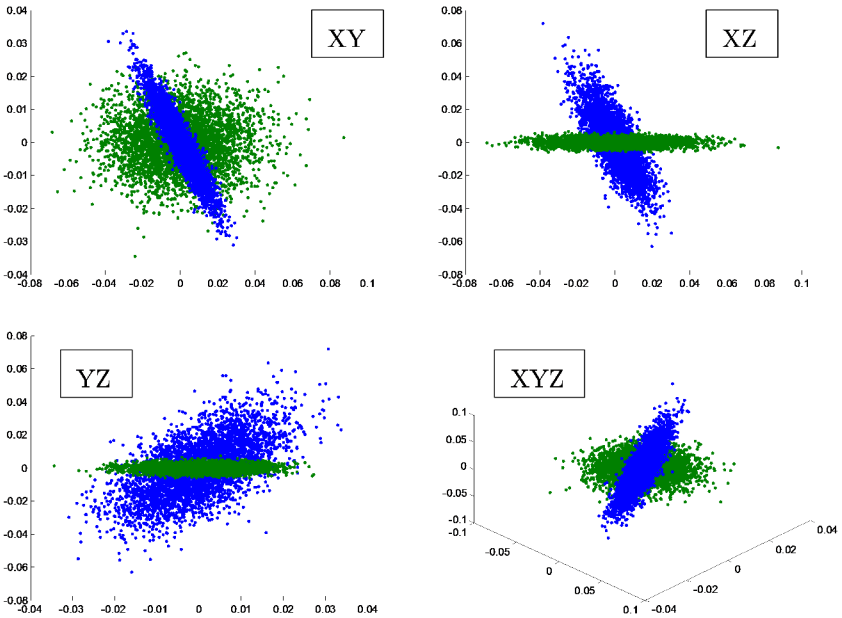
$$\boldsymbol{A}(:,j) = \boldsymbol{X}(:,j) - \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{X}(i,j)\,, \qquad j = 1..n \tag{4.16}$$

PCA jest n-wymiarową transformacją liniową danych $\mathbf{A}$ do ortogonalnego układu współrzędnych $\mathbf{W}$, w którym pierwsza oś określa kierunek największej wariancji (wrażliwości) danych, druga oś - kierunek prostopadły, maksymalizujący wariancję, itd. Współrzędne $\mathbf{T}$ danych w nowym układzie:

$$\begin{gathered} \boldsymbol{T} = \boldsymbol{A}\boldsymbol{W} \\ \boldsymbol{A} = \boldsymbol{T}\boldsymbol{W}^{\boldsymbol{T}} \\ \boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A} = \boldsymbol{W}\boldsymbol{T}^{\boldsymbol{T}}\boldsymbol{T}\boldsymbol{W}^{\boldsymbol{T}} \end{gathered} \tag{4.17}$$

Jeśli macierz kowariancji $\boldsymbol{T}^{\boldsymbol{T}}\boldsymbol{T}$ jest diagonalna (brak korelacji pomiędzy parametrami w nowym układzie współrzędnych), wówczas macierz $\mathbf{W}$ jest ortogonalną macierzą wektorów własnych macierzy kowariancji $\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}$. Metoda PCA związana jest zatem z problemem wyznaczenia wektorów własnych macierzy kowariancji danych. Interpretacją geometryczną takiej analizy jest wyznaczenie wielowymiarowej elipsoidy o kierunkach osi zgodnych z wektorami własnymi i o długościach osi równych wartościom własnym macierzy kowariancji danych. PCA określa transformację obrotu tej elipsoidy, po której jej osie, pokrywają się z wersorami układu współrzędnych.

Rys. 34 obrazuje tę sytuację w układzie współrzędnych czynników głównych dla danych o rozkładzie normalnym w trzech podstawowych kierunkach.



Rys. 34. Dane o rozkładzie normalnym w trzech podstawowych kierunkach, przed transformacją PCA (niebieski) i po (zielony) w rzutach na płaszczyzny układu czynników głównych

Wariancja rzutów obserwacji w kierunku v jest określona jako:

$$\|Av\|^2 = v^T A^T A v \tag{4.18}$$

Wyrażenie (4.19) dla jednostkowego wektora v określa iloraz Rayleigha, który przyjmuje maksymalną wartość dla największej wartości własnej macierzy kowariancji $\boldsymbol{A^T A}$. Wektor v jest wówczas dominującym wektorem własnym tej macierzy. Po wyznaczeniu wektora własnego v można zmodyfikować macierz $\boldsymbol{A}$ odejmując czynnik Avv^T związany z projekcją danych na ten wektor własny i poddać ponownie analizie zmodyfikowaną macierz $\boldsymbol{A}$ wyznaczając kolejną parę wartości i wektora własnego.

$$A_k = A_{k-1} - A_{k-1} v v^T$$

Błędy wyznaczania kolejnych par wartości i wektorów własnych tą metodą kumulują się. W przypadku konieczności wyznaczenia wszystkich par wartości i wektorów własnych, ze względu na mniejszą złożoność obliczeniową (niejawne tworzenie macierzy kowariancji) i większą dokładność, praktyczniejszą metodą jest faktoryzacja SVD macierzy $\boldsymbol{A}$ [40].

4.3.1 Faktoryzacja macierzy SVD

Dla dowolnej macierzy A możliwa jest faktoryzacja SVD

$$\boldsymbol{A = U \Sigma V^T} \tag{4.19}$$

w której macierz U jest ortogonalną macierzą lewych wektorów singularnych, V jest ortogonalną macierzą prawych wektorów singularnych, a macierz Λ jest diagonalną macierzą wartości singularnych. Ponieważ:

$$\boldsymbol{AA^T} = \boldsymbol{U\Sigma V^T V\Sigma U^T} = \boldsymbol{U\Sigma^2 U^T} \tag{4.20}$$

Macierz U jest macierzą wektorów własnych symetrycznej macierzy AA^T.
Macierz diagonalna Σ^2 jest macierzą wartości własnych macierzy AA^T

$$\boldsymbol{A^T A} = \boldsymbol{V\Sigma U^T U\Sigma V^T} = \boldsymbol{V\Sigma^2 V^T} \tag{4.21}$$

Macierz V jest macierzą wektorów własnych symetrycznej macierzy A^TA.
Macierz diagonalna Σ^2 jest macierzą wartości własnych macierzy A^TA

Diagonalizację wystarczy przeprowadzić tylko dla jednej z tych macierzy. Znając np. macierz V i Σ, macierz U można wyznaczyć z definicji faktoryzacji SVD (4.19):

$$\boldsymbol{U} = \boldsymbol{AV\Sigma^{-1}}$$

Faktoryzacja SVD umożliwia rozwiązywanie układów równań opisanych macierzami osobliwymi – nie posiadającymi macierzy odwrotnej. Możliwe jest wówczas wyznaczenie pseudoinwersji:

$$\boldsymbol{A^+} = \boldsymbol{V^T\Sigma^+ U} \tag{4.22}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \Rightarrow \boldsymbol{\Sigma^+} = \begin{bmatrix} 1/\sigma_1 & 0 & 0 \\ 0 & 1/\sigma_2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

W szczególności umożliwia ona rozwiązywanie problemu aproksymacji metodą najmniejszych kwadratów dla przypadków macierzy nieodwracalnych. Wartości σ są zawsze dodatnie ponieważ są kwadratami wartości własnych AA^T.

SVD znajduje jednak najszersze zastosowanie w redukcji wymiarowości problemów – metodzie czynników głównych PCA, umożliwiającej pomijanie kierunków o małych współczynnikach istotności. Jest ona standardową metodą wyznaczania komponentów PCA, ponieważ nie wymaga jawnego tworzenia macierzy kowariancji, które dla dużych i gęstych macierzy jest złożone obliczeniowo. Jeśli macierz $\boldsymbol{A}$ jest wycentrowana poprzez odjęcie wartości średnich w kolumnach macierzy obserwacji, produkt $\boldsymbol{A^TA}$ jest macierzą kowariancji danych. Prawe wektory singularne V macierzy $\boldsymbol{A}$ są wówczas wektorami ładunków komponentów głównych, czyli wektorami własnych macierzy kowariancji.

Praktyczna metoda wyznaczania dekompozycji SVD przebiega najczęściej dwuetapowo (algorytm, którego autorami są Gene Golub i Wiliam Kahan [41]). Pierwszym etapem jest wyznaczenie za pomocą odbić Householdera macierzy

trójkątnej lub bidiagonalnej, zachowującej podobieństwo macierzy. Drugi etap jest iteracyjną metodą numeryczną, która redukuje macierz wartości własnych do postaci diagonalnej przy użyciu różnych odmian algorytmu QR (np. przez ściganie zera [42]). Warto podkreślić, iż, mimo że drugi etap jest iteracyjny, to jego szybka zbieżność powoduje, że jest on mniej złożony obliczeniowo niż $O(n^3)$ pierwszego etapu.

Najnowsze i najszybsze algorytmy wyznaczania dekompozycji SVD oparte są na zasadzie „dziel i zwyciężaj", sprowadzając problem do rekurencyjnego znajdowania dekompozycji macierzy o 2-krotnie mniejszej liczbie kolumn i wierszy [43].

4.3.2 Odbicia Householdera i bidiagonalizacja

Transformacja Householdera H jest liniową operacją odbicia punktu względem płaszczyzny w przestrzeni n-wymiarowej. Płaszczyzna ta może być określona przez jednostkowy wektor do niej prostopadły (ortonormalny) v, wówczas:

$$H = I - 2vv^T \tag{4.23}$$

Macierz Householdera jest ortonormalna i symetryczna. Wynika z tego m.in., że macierzą do niej odwrotną jest ona sama. W dekompozycji SVD jest ona używana do bidiagonalizacji macierzy. Odbicia są stosowane naprzemiennie z lewej i prawej strony macierzy, redukując do zera odpowiednio elementy poniżej przekątnej i po prawej stronie pierwszej nadprzekątnej macierzy. Dekompozycja SVD nie wymaga stosowania transformacji zachowujących podobieństwo macierzy, dlatego obroty z lewej i prawej strony nie muszą być podobne i wynikowa macierz zawiera jedynie dwie przekątne [44].

Przedstawiony poniżej sposób wyznaczania macierzy Householdera jest chyba nieco prostszy od popularnej metody Atkinsona. Bardzo podobny algorytm można znaleźć w pracy Wilkinsona [36]. Szukana macierz sprowadza dany wektor d do wektora o kierunku pierwszego wersora, tzn. zeruje wszystkie współrzędne wektora Hd oprócz pierwszej (zachowując jego długość):

$$Hd = (I - 2vv^T)d = [\pm\|d\|, 0, \ldots, 0]^T \tag{4.24}$$

Ze względu na mniejszy błąd zaokrągleń, wybierany najczęściej jest znak przeciwny do pierwszej współrzędnej wektora d. Określając długość wektora d ze znakiem d_1 jako α:

$$\alpha = \ sign(d_1)\|d\|,$$

otrzymamy zależność:

$$2vv^T d = [d_1 + \alpha, d_2, \ldots, d_n]^T = w$$

ponieważ:

$$ww^T = 4vv^T dd^T vv^T = 4(v^T d)(d^T v)vv^T$$

$$w^T w = 4d^T vv^T vv^T d = 4(d^T v)(v^T d)$$

$$vv^T = \frac{ww^T}{w^T w} = \frac{ww^T}{\|w\|^2},$$

zatem:

$$H = I - 2\frac{ww^T}{\|w\|^2} = I - \frac{ww^T}{\alpha w_1} \tag{4.25}$$

Przyjmując za wektor d część pewnego wektora b poniżej indeksu k i używając macierzy Householdera H, można wyzerować wszystkie współrzędne wektora Hb, o indeksie mniejszym od k :

```
function H = householder(b, k)
% Constructs a matrix H that zeros entries in the product H*b below index k
n = length(b);
H = eye(n);
v(k:n,1) = b(k:n);
alpha = sign(v(k)) * norm(v);
if alpha ~= 0
    v(k) = v(k) + alpha;
    H = H - v * v' / (alpha * v(k));
end
```

Przetwarzając kolejno kolumny poniżej przekątnej i wiersze po prawej stronie nadprzekątnej można przeprowadzić bidiagonalizację dowolnej macierzy

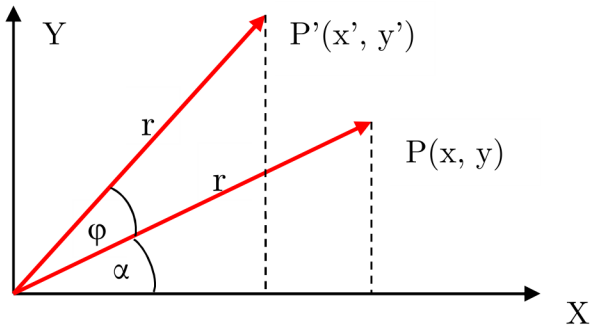
```
function [U,B,V] = bidiag(A)
% Algorithm 6.5-1 in Golub & Van Loan, "Matrix Computations"
% Finds U*B*V = A with U,V orthogonal and B upper bidiagonal
[m,n] = size(A);
B = A;
U = eye(m);
V = eye(n);
for k=1:n
    % eliminate non-zeros below the diagonal
    H = householder(B(:,k),k);
    U = U*H;
    B = H*B;
    % eliminate non-zeros to the right of the
    % superdiagonal by working with the transpose
    if k < n-1
        H = householder(B(k,:),k+1);
        B = B*H;
        V = V*H;
    end
end
```

Macierz bidiagonalna jest sprowadzana do diagonalnej przy użyciu algorytmu QR z przesunięciem.

4.3.3 Metoda Jacobiego

Dużo starszą metodą wyznaczania faktoryzacji SVD dowolnej macierzy A [45], ale nadal popularną ze względu na swoją prostotę i możliwość zrównoleglenia obliczeń jest przekształcenie jednostronne w metodzie Jacobiego. Wartości singularne w tej

metodzie są również wyznaczane z wyższą dokładnością w stosunku do innych metod [46] [47]. Przekształcenie jednostronne wyznacza niejawnie symetryczną macierz $\Sigma = A^T A$ i za pomocą sekwencji obrotów Jacobiego (nazywanych również obrotami Givensa), zachowując podobieństwo, sprowadza ją do macierzy diagonalnej.



Rys. 35. Obrót punktu na płaszczyźnie o kąt φ

Zapisując biegunowo punkt P, oraz jego obraz P' po obrocie o kąt φ (Rys. 35)

$$\mathrm{P} = (r\cos\alpha\,, r\sin\alpha)$$

$$\mathrm{P}' = (r\cos(\alpha+\varphi)\,, r\sin(\alpha+\varphi))$$

można wyznaczyć współrzędne punktu P' na podstawie P i kąta φ

$$\begin{cases} x' = r\cos(\alpha+\varphi) = r\cos\alpha\cos\varphi - r\sin\alpha\sin\varphi = x\cos\varphi - y\sin\varphi \\ y' = r\sin(\alpha+\varphi) = r\sin\alpha\cos\varphi + r\cos\alpha\sin\varphi = y\cos\varphi + x\sin\varphi \end{cases}$$

Obrót można zatem przedstawić w postaci macierzowej:

$$P' = P * \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix} = P * \boldsymbol{J}_{X,Y} \tag{4.26}$$

W przestrzeni n-wymiarowej, macierz obrotu na płaszczyźnie określonej przez wersory osi (i, j) przybiera analogiczną postać macierzy jednostkowej, za wyjątkiem czterech elementów na przecięciu wierszy i kolumn i, j :

$$\boldsymbol{J}_{i,j} = \begin{array}{c} \\ i \\ \\ j \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} & & i & & j \end{array} \\ \begin{bmatrix} \ldots 1 \ldots & & & & \\ & \cos\varphi & & \sin\varphi & \\ & & \ldots 1 \ldots & & \\ & -\sin\varphi & & \cos\varphi & \\ & & & & \ldots 1 \ldots \end{bmatrix} \end{array}$$

W metodzie Jacobiego, transformacje wykonywane w każdej iteracji są przekształceniami zachowującymi podobieństwo macierzy, tzn.:

$$\Sigma_{k+1} = J_{i,j}^T * \Sigma_k * J_{i,j} \tag{4.27}$$

Elementy macierzy Σ na przecięciu wierszy i kolumn i, j wyznaczane są na podstawie iloczynu skalarnego kolumn macierzy A (ze względu na złożenie macierzy A^TA). Algorytm dobiera kąt obrotu na kolejnych płaszczyznach tak, by wyzerować elementy ${\sigma_{ij}}', {\sigma_{ji}}'$ znajdujące się poza przekątną. Przy tym $\sigma_{ji} = \sigma_{ij}$, ponieważ macierz jest symetryczna.

$$\begin{bmatrix} {\sigma_{ii}}' & 0 \\ 0 & {\sigma_{jj}}' \end{bmatrix} = \begin{bmatrix} cos\varphi & -sin\varphi \\ sin\varphi & cos\varphi \end{bmatrix} \begin{bmatrix} \sigma_{ii} & \sigma_{ij} \\ \sigma_{ij} & \sigma_{jj} \end{bmatrix} \begin{bmatrix} cos\varphi & sin\varphi \\ -sin\varphi & cos\varphi \end{bmatrix}$$

Wynika stąd warunek (por. [48]):

$${\sigma_{ij}}' = {\sigma_{ji}}' = (cos\varphi\sigma_{ii} - sin\varphi\sigma_{ij})sin\varphi + (cos\varphi\sigma_{ij} - sin\varphi\sigma_{jj})cos\varphi = 0$$

$$\frac{\sigma_{jj} - \sigma_{ii}}{\sigma_{ij}} = \frac{cos^2\varphi - sin^2\varphi}{sin\varphi cos\varphi} = 2ctg(2\varphi) \tag{4.28}$$

$$ctg^2\varphi - 1 = 2ctg(2\varphi)ctg\varphi$$

$$ctg\varphi = ctg(2\varphi) \pm \sqrt{1 + ctg^2(2\varphi)} \tag{4.29}$$

Ze względu na mniejszy błąd numeryczny, w równaniu (4.29) przyjęty został znak $ctg(2\varphi)$, oraz:

$$cos\varphi = \frac{1}{\sqrt{1 + ctg^{-2}\varphi}}$$

$$sin\varphi = \frac{cos\varphi}{ctg\varphi}$$

Obroty zmieniają cały wiersz i-ty i kolumnę j-tą w macierzy Σ, dlatego obroty w kolejnych iteracjach zastępują uzyskane w poprzednich iteracjach miejsca zerowe. Okazuje się jednak, że proces jest szybko zbieżny do macierzy diagonalnej. Dla wyboru kolejnych płaszczyzn (i, j) obrotów zgodnym z przeglądaniem macierzy wierszami, zbieżność ta jest kwadratowa.

Warunek zatrzymania iteracji w poniższym algorytmie jest w zasadzie zgodny z warunkiem podanym przez Demela i Veseliča [46], ale badany jest kwadrat błędu, dzięki czemu unika się liczenia pierwiastka kwadratowego w wewnętrznej pętli. Obroty wykonywane są tylko wówczas, jeśli elementy poza przekątną są większe od założonego błędu. Wektory kolumn macierzy powstałej w wyniku transformacji macierzy A przez złożenie wszystkich obrotów, mają długość odpowiadającą jej wartościom singularnym. Ich unormowanie wyznacza lewe wektory singularne. Prawe wektory wyznaczane są na podstawie pseudoinwersji wartości singularnych, a nie klasycznie – poprzez złożenie obrotów Jacobiego. Przyspiesza to wyznaczenie wektorów singularnych, jeśli nie są istotne wektory singularne związane z bardzo małymi wartościami singularnymi.

```
function [U,S,V] = SVD(A)
% One-sided Jacobi algorithm for SVD
n = size(A,2);
U = A;
converge = eps + 1;
while converge > eps
    converge = 0;
    for j = 2:n
        for i = 1:j-1
            % compute [a,c; c,b] = (i,j) submatrix of U'*U
            c = U(:,i)' * U(:,j);
            if (abs(c) > eps)
                a = U(:,i)' * U(:,i);
                b = U(:,j)' * U(:,j);
                converge = max(converge, c/a*c/b);
                % compute Jacobi rotation that diagonalizes [a,c; c,b]
                ctan2L = (b-a)/(2*c);
                ctanL = ctan2L + sign(ctan2L) * sqrt(1 + ctan2L * ctan2L);
                cosL = 1 / sqrt(1 + 1 / (ctanL * ctanL));
                sinL = cosL / ctanL;
                tmp = U(:,i);
                U(:,i) = cosL * tmp - sinL * U(:,j);
                U(:,j) = sinL * tmp + cosL * U(:,j);
            end
        end
    end
end
% the singular values are the norms of the columns of U
Sinv = zeros(n,n);
for j = 1:n
    singvals(j) = norm(U(:,j));
    if (singvals(j) > eps)
        U(:,j) = U(:,j) / singvals(j);
        Sinv(j,j) = 1 / singvals(j);
    end;
end
S = diag(singvals);
V = A' * U * Sinv;
```


4.4 Segmentacja K-means (KM)

Najprostszą segmentacją jest segmentacja progowa, dzieląca zakres wartości pikseli na przedziały przynależące do obiektów. Bardziej wyrafinowane metody starają się interacyjnie poprawić ten podział. Najpopularniejszym takim algorytmem jest klastrowa segmentacja k-means [49] [50] [51] [52]. Sposób podziału obrazu na segmenty metodą k-means zależy jedynie od wartości piksela, nie uwzględniając ani jego położenia, ani wartości pikseli sąsiednich, dlatego jej wynikiem są w ogólności klastry niespójne. Liczba segmentów-klasterów k, na które obraz zostanie podzielony ustalana jest z góry. W pierwszej iteracji wybieranych jest losowo lub heurystycznie k wartości środków segmentów i piksele przydzielane są do najbliższych im w sensie odległości euklidesowej segmentów. Środki klasterów są następnie przeliczane do wartości średniej ze wszystkich pikseli należących do danego klastra i procedura przydziału jest przeprowadzana ponownie. Algorytm w każdej iteracji wykonuje zatem dwie operacje: przypisania pikseli do klasterów i korekcji wartości środków klasterów. Nietypową cechą k-means wśród algorytmów segmentacji jest możliwość

dokładnego określenia błędu. Algorytm k-means minimalizuje sumę kwadratów odległości wartości pikseli od przyporządkowanych im środków segmentów.

$$error = \sum_{i=1}^{P} \min_{j \in <1,S>} (x[i] - c[j])^2 \tag{4.30}$$

Algorytm k-means jest zawsze zbieżny, gdyż średnia arytmetyczna jest estymatą błędu średniokwadratowego, ale iteracje można również przerwać w momencie uzyskania wystarczająco małego błędu segmentacji. Jeśli funkcją kosztu byłaby suma wartości bezwzględnych odległości pikseli od przypisanych środków, w drugim kroku każdej iteracji należałoby korygować środki segmentów do wartości mediany z pikseli należących do klastra (algorytm k-median).

Algorytm k-means można również wykorzystać do klasteryzacji danych wielowymiarowych [53] [54]. Procedura segmentacji sekwencji obrazów ma za zadanie wyznaczyć zbiory składające się z podobnych charakterystyk wartości pikseli w funkcji numeru kadru. Metoda k-means porównuje ze sobą wektory wartości pikseli w kolejnych kadrach, czyli charakterystyki pikseli. Po segmentacji, dla każdego klastra, otrzymujemy jego środek w postaci wektora – średniej charakterystyki klastra w funkcji kadru. Dla sekwencji obrazów definiujemy błąd bezwzględny segmentacji na S obiektów jako sumę odchyleń wartości wszystkich P pikseli od przyporządkowanych im środków segmentów dla wszystkich K kadrów:

$$error = \sum_{i=1}^{P} \min_{j=1..S} \sum_{k=1}^{K} (x[k,i] - c[k,j])^2 = \sum_{i=1}^{P} \min_{j=1..S} \|x[i] - c[j]\|^2 \tag{4.31}$$

Warto podkreślić, że algorytm k-means łatwo zaimplementować z wykorzystaniem przetwarzania równoległego pikseli w celu przyspieszenia obliczeń.

Algorytm k-means jest szczególnym przypadkiem segmentacji FCM (Fuzzy C-Means) [55], w której zakłada się, że piksele mogą należeć do segmentów zgodnie z określoną funkcją wagową μ. Zakładając, że S oznacza liczbę segmentów a r stopień rozmycia przynależności pikseli do segmentów:

$$\mu(x[i], c[j]) = \frac{\|x[i] - c[j]\|^{-2/(r-1)}}{\sum_{k=1}^{S} \|x[i] - c[k]\|^{-2/(r-1)}} \tag{4.32}$$

W każdej iteracji środki klasterów modyfikowane są przez wyznaczenie środka ciężkości segmentów:

$$c[j] = \frac{\sum_{i=1}^{P} \mu(x[i], c[j]) x[i]}{\sum_{i=1}^{P} \mu(x[i], c[j])} \tag{4.33}$$

Jeśli tylko dla jednego segmentu waga jest niezerowa, algorytm degeneruje się do metody k-means. Algorytm FCM w ogólności znajduje lepsze rozwiązanie, jest jednak

znacznie bardziej złożony obliczeniowo, jest trudniej zbieżny i wymaga dobrania odpowiedniego współczynnika rozmycia r.

Algorytm k-means jest zawsze zbieżny jeśli miarą odległości wektorów jest metryka euklidesowa, ale o szybkości zbieżności i dokładności rozwiązania decyduje przede wszystkim wybór początkowy środków klasterów (Pena et al., 1999 [56]). Pomimo tak silnej zależności, standardowym podejściem jest wybór losowy środków, z jednakowym prawdopodobieństwem, spośród wartości pikseli źródłowych, tzw. inicjalizacja Forgy'ego. Znane są jednak inicjalizacje, które optymalizują początkowy rozkład środków, np. inicjalizacja Katsavounidisa KKZ oraz zdobywający coraz większą popularność algorytm kmeans++. Poniżej przedstawione zostały implementacje metod w klasie *TSegmenter*, które wyznaczają określoną liczbę *SegmentCount* środków przy użyciu najpopularniejszych sposobów inicjalizacji KM. Przedstawiona została również propozycja nowej metody inicjalizacji, opartej na oszacowaniu gęstości pikseli.

4.4.1 Inicjalizacja Forgy'ego

Inicjalizacja Forgy'ego zakłada przyjęcie za początkowe środki segmentów wartości losowych, wybranych spośród danych wejściowych, przy jednorodnym rozkładzie prawdopodobieństwa [49]. Wynik segmentacji może się znacznie różnić od minimum globalnego i najczęściej polega na wyborze najlepszego rozwiązania z kilku przeprowadzonych segmentacji. Wydłuża to znacznie czas potrzebny na przetwarzanie obrazu.

Implementacja zakłada, że na tym etapie wszystkie piksele podlegające segmentacji są dziećmi wspólnego rodzica – Root'a.

```
public TSegment[] FA(int centerCount)
{
    TSegment[] Centers = new TSegment[centerCount];
    Random random = new Random();
    for (int k = 0; k < centerCount; k++)
        Centers[k] = Root.Children[random.Next(Root.Children.Count)];
    return Centers;
}
```

4.4.2 Inicjalizacja KKZ

KKZ (Katsavounidis et al. [57]) – algorytm z 1997 roku, nazwany od inicjałów autorów, w którym na pierwszy środek można wybrać piksel o najmniejszej lub największej wartości. Kolejne przyjmują wartości pikseli położonych najdalej od znalezionych już środków. Maksymalizując odległość pomiędzy środkami gwarantuje się dobrą separację segmentów. Wadą tej metody jest częste tworzenie segmentów z niewielkiej liczby pikseli o wartościach skrajnych, tzw. outlierów.

W klasie *TSegmenter* metoda *KKZ* na pierwszy środek wybiera piksel o maksymalnej jasności. Implementacja metody może być bardzo szybka, ponieważ przechowując

tablicę minimalnych odległości każdego piksela od znalezionych już środków, wybieramy na kolejny środek, piksel o maksymalnej odległości (algorytm maxi-min). Korekcja tablicy odległości jest wymagana jedynie dla pikseli najbliższych znalezionemu środkowi.

```
public TSegment[] KKZ(int centerCount)
{
    TSegment[] Centers = new TSegment[centerCount];
    Centers[0] = Root.Children[0];
    for (int i = 0; i < Root.Children.Count; i++)
    {
        TSegment pixel = Root.Children[i];
        if (pixel.Intensity > Centers[0].Intensity)
            Centers[0] = pixel;
    }
    for (int k = 1; k < centerCount; k++)
    {
        double maxDist = 0;
        for (int i = 0; i < Root.Children.Count; i++)
        {
            TSegment pixel = Root.Children[i];
            if (k == 1) pixel.Distance = double.MaxValue;
            double d = pixel.DistanceTo(Centers[k - 1]);
            if (d < pixel.Distance)
                pixel.Distance = d;
            if (pixel.Distance > maxDist)
            {
                maxDist = pixel.Distance;
                Centers[k] = pixel;
            }
        }
    }
    return Centers;
}
```

4.4.3 Algorytm kmeans++

W 2007 roku D.Arthur i S.Vassilvitskii [58] przedstawili algorytm kmeans++, który statystycznie wykazuje szybszą zbieżność od zupełnie losowej inicjalizacji, a jednocześnie zapewnia separację początkowych środków segmentów. W algorytmie na początkowe środki przyjmuje się wartości losowe, wybrane spośród danych wejściowych, przy rozkładzie prawdopodobieństwa proporcjonalnym do kwadratu odległości od już wyznaczonych środków. Metoda łączy w sobie zatem wykorzystanie statystyki (jak FA) i analizę odległości od już wyznaczonych środków (tak, jak KKZ).

W implementacji metody KMPP, pierwszy środek wybierany jest losowo z wartości wszystkich pikseli, przy jednorodnym rozkładzie prawdopodobieństwa. Na kolejne środki wybierane są piksele z prawdopodobieństwem $\frac{D(x)^2}{\sum_n D(x_n)^2}$, gdzie D(x) określa minimalną odległość piksela o wartości x do zbioru znalezionych środków. W tym celu piksele sortowane są według odległości od znalezionych środków. Losowana jest pewna wartość *cumDist* z zakresu od 0 do sumy wszystkich odległości i znajdowany

jest indeks piksela i, dla którego suma kolejnych odległości pikseli przekracza wartość $cumDist$.

$$\sum_{n}^{i} D(x_n)^2 \geq cumDist > \sum_{n}^{i-1} D(x_n)^2$$

Znaleziony piksel jest dodawany do wektora środków.

```
public TSegment[] KMPP(int centerCount)
{
    TSegment[] Centers = new TSegment[centerCount];
    Random random = new Random();
    Centers[0] = Root.Children[random.Next(Root.Children.Count)];
    for (int k = 1; k < centerCount; k++)
    {
        double sumDist = 0;
        for (int i = 0; i < Root.Children.Count; i++)
        {
            TSegment pixel = Root.Children[i];
            if (k == 1) pixel.Distance = double.MaxValue;
            double d = pixel.DistanceTo(Centers[k - 1]);
            if (d < pixel.Distance)
                pixel.Distance = d;
            sumDist += pixel.Distance;
        }
        Root.Children.Sort((segA, segB) => segA.Distance.CompareTo(segB.Distance));
        double cumDist = random.NextDouble() * sumDist;
        sumDist = 0;
        for (int i = 0; i < Root.Children.Count; i++)
        {
            sumDist += Root.Children[i].Distance;
            if (sumDist >= cumDist)
            {
                Centers[k] = Root.Children[i];
                break;
            }
        }
    }
    return Centers;
}
```

4.4.4 Inicjalizacja uwzględniająca gęstość pikseli

Powyższe inicjalizacje algorytmu k-means nie uwzględniają jednak gęstości określonej jako rozkład wartości pikseli. Wymaga ona obliczenia wzajemnych odległości pomiędzy pikselami, czyli liczby operacji rzędu $O(n^2)$. Dla problemów wielowymiarowych w celu oszacowania gęstości można posłużyć się strukturą drzewa kd dzielącego przestrzeń danych w kolejnych wymiarach [59] [60]. O dużej gęstości piksela decydują niewielkie odległości od jego sąsiadów. Można tę funkcję określić np. poprzez sumowanie odwrotności odległości od pozostałych pikseli (Inverse Distance Weighting Function) :

$$\rho[k] = \sum_{i=1}^{P} \frac{1}{\|x[i] - x[k]\|^2 + \varepsilon} \tag{4.34}$$

Czynnik ε jest dodatni i związany z kwantyzacją danych, w wyniku której niektóre odległości pomiędzy pikselami są zerowe. Wyznaczenie tak zdefiniowanej funkcji jest bardzo kosztowne obliczeniowo, ale można jej wartość poddać oszacowaniu. Jeśli przyjąć, że odległości piksela k od pozostałych są zawsze takie same, wówczas:

$$\rho[k] = \sum_{i=1}^{P} \frac{1}{\|x[i] - x[k]\|^2 + \varepsilon} \cong \frac{P}{d[k] + \varepsilon} \tag{4.35}$$

Gęstość piksela k można określić korzystając z rozwinięcia w szereg Taylora funkcji $(d+1)^{-1} = 1 - d + d^2 - d^3 + \cdots$, ponieważ:

$$\rho[k] = \sum_{i=1}^{P} \frac{1}{\|x[i] - x[k]\|^2 + \varepsilon} = \frac{1}{\varepsilon} \sum_{i=1}^{P} \left(\frac{\|x[i] - x[k]\|^2}{\varepsilon} + 1 \right)^{-1} \tag{4.36}$$

Rozwijając również prawą stronę przybliżenia (4.35), otrzymujemy:

$$\begin{aligned} &\frac{1}{\varepsilon} \left(P - \frac{1}{\varepsilon} \sum_{i=1}^{P} \|x[i] - x[k]\|^2 + \frac{1}{\varepsilon^2} \sum_{i=1}^{P} \|x[i] - x[k]\|^4 - \cdots \right) \\ &\qquad \cong \frac{P}{\varepsilon} \left(1 - \frac{d[k]}{\varepsilon} + \left(\frac{d[k]}{\varepsilon} \right)^2 - \cdots \right) \end{aligned} \tag{4.37}$$

Z porównania pierwszych składników rozwinięć wynika, że odległość d dla piksela k powinna być równa średniej ze wszystkich odległości tego piksela względem pozostałych. Można ją szybko wyznaczyć dla wszystkich pikseli na podstawie wartości średniej $\bar{x}$ i wariancji Var[x], przy założeniu jednakowego prawdopodobieństwa przyjmowanych wartości przez piksele.

$$\begin{aligned} d[k] &= \frac{1}{P} \sum_{i=1}^{P} \| \; x[i] - x[k] \; \|^2 = \\ &= \frac{1}{P} \sum_{i=1}^{P} \|x[i] - \bar{x}\|^2 + \|\bar{x} - x[k]\|^2 = Var[x] + \|\bar{x} - x[k]\|^2 \end{aligned} \tag{4.38}$$

Przybliżenie będzie prawdziwe jeśli ciągi nie będą się rozbiegać zbyt szybko, tzn. dla $\varepsilon \geq d[k]$. Ponieważ czynnik ε powinien być jak najmniejszy i stały dla wszystkich pikseli, przyjęto za jego wartość średnią ze wszystkich odległości d[k]

$$\varepsilon = \frac{1}{P} \sum_{i=1}^{P} d[k] = \frac{1}{P} \sum_{i=1}^{P} (Var[x] + \|\bar{x} - x[k]\|^2) = 2 * Var[x] \tag{4.39}$$

Gęstość piksela można zatem oszacować jako [61]:

$$\rho[k] \cong \frac{P}{3 * Var[x] + \|\bar{x} - x[k]\|^2} \tag{4.40}$$

Metoda inicjalizacji uwzględniającą gęstość pikseli określona została jako High Density (HD). Na środki segmentów powinny zostać wybrane piksele o jak największej gęstości ale i jak najbardziej odległe od już wyznaczonych środków. Dlatego jako pierwszy środek jest wybierany piksel o największej gęstości, natomiast pozostałe środki segmentów przyjmują wartości pikseli o maksymalnym iloczynie gęstości piksela i jego odległości od już wyznaczonych środków.

```
public TSegment[] HD(int centerCount)
{
    TSegment[] Centers = new TSegment[centerCount];
    double variance = 0;
    for (int i = 0; i < Root.Children.Count; i++)
        variance += Root.Children[i].DistanceTo(Root);
    variance /= Root.Children.Count;
    double[] densities = new double[Root.Children.Count];
    double maxDensity = -1;
    for (int i = 0; i < Root.Children.Count; i++)
    {
        TSegment pixel = Root.Children[i];
        densities[i] = 1 / (3 * variance + pixel.DistanceTo(Root));
        if (densities[i] > maxDensity)
        {
            maxDensity = densities[i];
            Centers[0] = pixel;
        }
    }
    for (int k = 1; k < centerCount; k++)
    {
        double maxDist = 0;
        for (int i = 0; i < Root.Children.Count; i++)
        {
            TSegment pixel = Root.Children[i];
            if (k == 1) pixel.Distance = double.MaxValue;
            double d = pixel.DistanceTo(Centers[k - 1]);
            if (d < pixel.Distance)
                pixel.Distance = d;
            double len = densities[i] * pixel.Distance;
            if (len > maxDist)
            {
                maxDist = len;
                Centers[k] = pixel;
            }
        }
    }
    return Centers;
}
```

Implementacja algorytmu k-means korzysta z inicjalizacji HD. Może ona zostać zastąpiona również przez inne metody inicjalizacji. Pikselom wybranym jako początkowe środki segmentów przypisywany jest nowy rodzic, będący dzieckiem Root'a. Podczas przypisywania rodzica, kolor piksela jest automatycznie dodawany do wartości rodzica w setterze property Parent, zatem przy przypisaniu pojedynczego piksela obie wartości są takie same. Pozostałe piksele są wpisywane do listy dzieci pierwszego rodzica. W ten sposób rodzicami pikseli są segmenty o początkowych wartościach środków zgodnych z inicjalizacją. W każdej iteracji tworzone jest nowe

drzewo hierarchii segmentów i piksele są na nowo przydzielane do skorygowanych środków. Nowe drzewo zastępuje stare. Obliczenia są przyspieszane dla kolejnych iteracji poprzez śledzenie segmentów, które nie uległy zmianie i nie wymagają ponownego obliczania odległości pikseli do środków (w tablicach *wasChanged* - poprzednia iteracja i *isChanged* - aktualna). Piksele przetwarzane są równolegle dzięki pętli *Parallel.For*, z lockiem na przypisaniu nowego rodzica (*List* nie jest klasą thread-safe we frameworku .NET 4.0)

```
public void KMeans(int count)
{
    TSegment[] centers = HD(count);
    TRootSegment newRoot = new TRootSegment(this);
    newRoot.Children.Capacity = count;
    TSegment parent;
    for (int i = 0; i < count; i++)
    {
        parent = new TSegment();
        parent.Parent = newRoot;
        centers[i].Parent = parent;
    }
    for (int i = 0; i < Root.Children.Count; i++)
    {
        TSegment seg = Root.Children[i];
        if (seg.Parent != Root) continue;
        newRoot.Children[0].Children.Add(seg);
    }
    bool[] wasChanged = new bool[count];
    for (int i = 0; i < count; i++)
        wasChanged[i] = true;
    bool converged = false;
    int runs = 0;
    while (!converged && runs < 20)
    {
        converged = true;
        double error = 0;
        runs++;
        Root = newRoot;
        newRoot = new TRootSegment(this);
        bool[] isChanged = new bool[count];
        for (int i = 0; i < count; i++)
        {
            TSegment seg;
            if (!wasChanged[i])
                seg = Root.Children[i];
            else
                seg = new TSegment();
            seg.Parent = newRoot;
        }
        Parallel.For(0, Root.Children.Count, i =>
        {
            if (wasChanged[i])
            {
                TSegment seg = Root.Children[i];
                int idx = 0;
                for (int j = 0; j < seg.Children.Count; j++)
                {
                    double minDist = 1;
                    TSegment child = seg.Children[j];
```

```
                    for (int k = 0; k < Root.Children.Count; k++)
                    {
                        double d = child.DistanceTo(Root.Children[k]);
                        if (d < minDist)
                        {
                            minDist = d;
                            idx = k;
                        }
                    }
                    lock (newRoot)
                    {
                        error += minDist;
                        child.Parent = newRoot.Children[idx];
                    }
                    if (idx != i)
                    {
                        isChanged[i] = true;
                        isChanged[idx] = true;
                        converged = false;
                    }
                }
            }
        });
        isChanged.CopyTo(wasChanged, 0);
    }
    FDest = null;
}
```

4.5 Segmentacja wododziałowa - definicja zalewowa

W segmentacji wododziałowej opartej na definicji zalewowej przetwarzany jest tzw. obraz gradientowy. Gradient informuje o kierunku największej zmiany wartości piksela obrazu źródłowego względem jego sąsiedztwa, natomiast sam obraz gradientowy powstaje przez przypisanie do pikseli wartości modułu gradientu, a więc wartości największej odległości pomiędzy pikselem i pikselami z jego sąsiedztwa. Obraz gradientowy interpretowany jest jako mapa topograficzna, w której wartość piksela oznacza wysokość (obraz jako wykres funkcji 3D). Minima lokalne wyznaczają piksele najbardziej podobne do swoich sąsiadów. Stanowią one środki obiektów-basenów. Podział segmentacyjny uzyskiwany jest poprzez analogię zalewania mapy wodą od poziomu najniższego do najwyższego i budowania basenów poprzez dołączanie przyległych do nich pikseli. W miejscach, gdzie woda z różnych basenów łączy się, stawiane są tamy (na krawędziach obiektów) – stąd nazwa algorytmu. Procedura została zastosowana po raz pierwszy przez Digabel i Lantuéjoul [62]. Przytoczony poniżej algorytm jest jednym z najpopularniejszych i został przedstawiony przez S. Beucher i F. Meyer'a w 1993 roku [63]

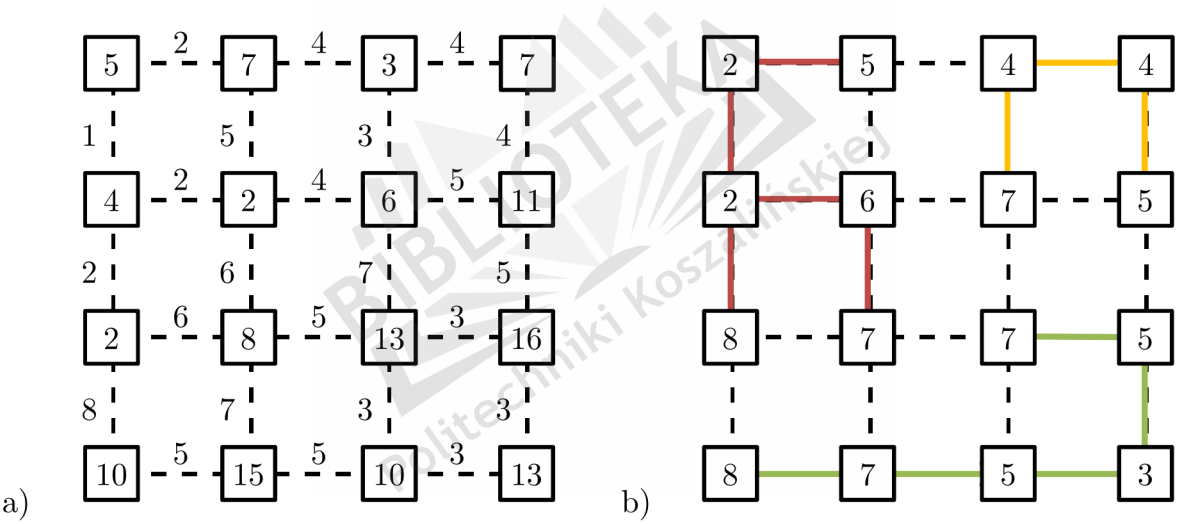
Algorytm zalewowy Meyera

1. Każdemu minimum lokalnemu nadaj inną etykietę i wstaw do zbioru S

2. Wyciągnij ze zbioru S piksel x o minimalnej wysokości. Przypisz etykietę piksela x do wszystkich pikseli y, które nie mają etykiety a są przyległe do x. Wstaw y do S
3. Powtarzaj krok 2, aż S będzie pusty

Pomimo zwięzłości zapisu, algorytm jest mało efektywny w implementacji. Konieczne jest wcześniejsze wyznaczenie minimów lokalnych, a także szukanie wartości minimalnych w zbiorze S w każdej iteracji pętli.

Ilustracja algorytmu zalewowego segmentacji wododziałowej dla obrazu monochromatycznego została przedstawiona na Rys. 36. Na podstawie obrazu źródłowego, zakładając sąsiedztwo 4-pikselowe, tworzony jest obraz gradientowy, którego piksele przyjmują wartość największej różnicy wartości pikseli obrazu źródłowego względem jego sąsiadów. Na obrazie gradientowym zaznaczono piksele dołączone do minimów (o wartościach 2, 3 i 4), poprzez wykonanie algorytmu zalewowego.

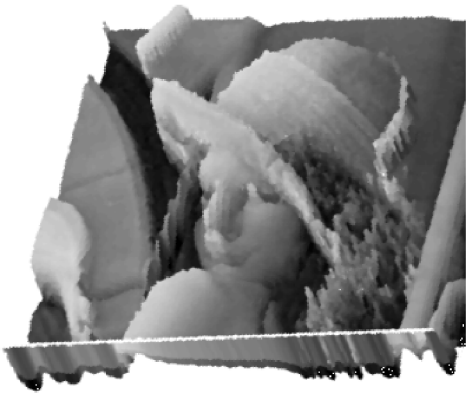


Rys. 36. Obraz źródłowy z różnicami bezwzględnymi pomiędzy pikselami (a) oraz obraz gradientowy, którego piksele mają wartość największego gradientu, z zaznaczonymi basenami (b).

4.6 Segmentacja wododziałowa - definicja topograficzna

Segmentację wododziałową można również zdefiniować poprzez interpretację wartości pikseli obrazu jako wysokości topograficznych (Rys. 37), oraz wykorzystanie tzw. zasady spadającej kropli, która spływa z piksela obrazu w kierunku lokalnego minimum. Wszystkie piksele, przez które płyną strumienie utworzone z kropel spływających do tego samego minimum tworzą zlewisko wspólnego basenu (segment). Zgodnie z tą zasadą, do piksela dołączany jest piksel sąsiedni, który najmniej różni się od badanego piksela, a następnie rekurencyjnie badany jest nowo dołączony piksel. Piksele, z których krople mogą spływać do co najmniej dwóch basenów, tworzą

grzbiety, czyli krawędzie segmentów. Liczba znalezionych minimów określa liczbę basenów i przy braku innych założeń jest przeważnie zbyt duża. Problem nadsegmentacji rozwiązuje się najczęściej poprzez kwantyzację możliwych wartości odległości pomiędzy pikselami.



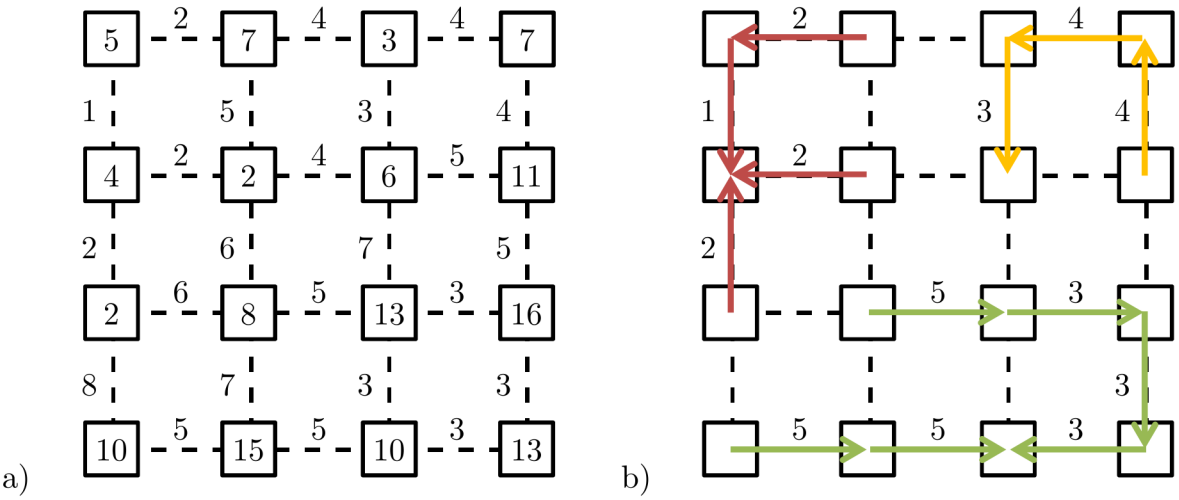
Rys. 37. Topograficzna interpretacja wartości pikseli w obrazie Lena

Segmentację wododziałową można przedstawić w postaci grafu ważonego krawędziowo, w którym wierzchołkami są piksele obrazu a wagami krawędzi jest odległość pomiędzy pikselami. Minimalny las spinający (MSF) tworzą strumienie płynące z pikseli do lokalnego minimum poprzez wybieranie krawędzi z najmniejszą wagą. MSF spina wszystkie wierzchołki grafu. Każde drzewo (podgraf) zawiera minimum i stanowi odrębny obiekt (Rys. 38). W 2009 roku grupa francuskich naukowców z Katedry Informatyki na Paryskim Uniwersytecie Marne-la-Vallée: J.Cousty, G.Bertrand, L.Najman i M.Couprie [64] opublikowała algorytm segmentacji wododziałowej oparty na definicji topograficznej, w której na grafie ważonym krawędziowo, wyznaczany jest optymalny las spinający (OSF).

Algorytm Optymalnego Lasu Spinającego (Optimal Spanning Forest) :

1. Wybierany jest kolejny piksel obrazu źródłowego bez etykiety. Traktowany jest on jako źródło strumienia.
2. Do strumienia dodawane są piksele wybierane zgodnie z kierunkiem najmniejszego gradientu aż do osiągnięcia minimum lokalnego lub innego strumienia. Jeśli istnieje kilka węzłów o takiej samej wartości minimalnej gradientu, tworzone jest rozlewisko (eksploracja wszerz), aż do znalezienia wartości mniejszej - ujścia (powrót do eksploracji w głąb).
3. Jeśli strumień napotka inny strumień, przejmuje jego etykietę, jeśli nie – nadawana jest mu kolejna etykieta.
4. Powtarzaj krok 1, aż do przetworzenia wszystkich pikseli

Algorytm ten jest efektywny obliczeniowo, z quasi-liniową zależnością obliczeniową od wielkości obrazu



Rys. 38. Ilustracja algorytmu topograficznego OSF segmentacji wododziałowej dla obrazu monochromatycznego. Obraz źródłowy z różnicami bezwzględnymi pomiędzy pikselami (a) oraz obraz gradientowy, na którym zaznaczono strumienie spływające do tych samych minimów i tworzące segmenty (b).

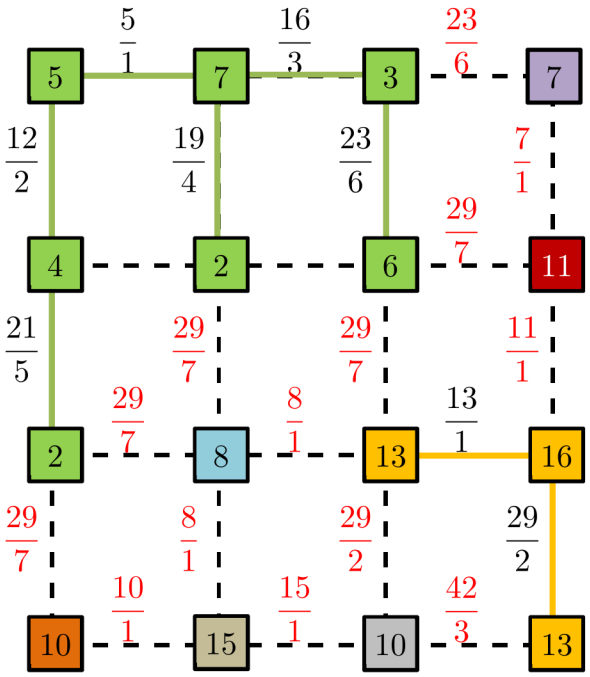
```
public TSegment Flow()
{
    TSegmenter owner = Owner;
    TSegment refSeg = null;
    double actGrad = Double.MaxValue;
    for (int actIndex = 0; actIndex < Parent.Children.Count; actIndex++)
    {
        refSeg = Parent.Children[actIndex];
        double minGrad = Double.MaxValue;
        List<int> sw = new List<int>();
        for (int j = 0; j < refSeg.Neighbors.Count; j++)
        {
            TSegment curSeg = refSeg.Neighbors[j];
            if (curSeg.Parent != Parent)
            {
                double grad = curSeg.DistanceTo(refSeg);
                //grad = Math.Floor(50 * grad) / 50;
                if (grad < minGrad)
                {
                    sw.Clear();
                    minGrad = grad;
                }
                if (grad == minGrad) sw.Add(j);
                if (curSeg.Parent != owner.Root)
                    Parent.AddNeighbor(curSeg.Parent);
            }
        }
        if (minGrad <= actGrad)
        {
            actGrad = minGrad;
            for (int k = 0; k < sw.Count; k++)
            {
                TSegment curSeg = refSeg.Neighbors[sw[k]];
                if (curSeg.Parent == owner.Root)
                    curSeg.Parent = Parent;
                else
                {
                    TSegment oldParent = Parent;
                    for (int u = oldParent.Children.Count - 1; u >= 0; u--)
```

```
                            oldParent.Children[u].Parent = curSeg.Parent;
                        oldParent.Children.Clear();
                        actIndex = Parent.Children.Count;
                        break;
                    }
                }
            }
        }
        return refSeg;
    }
```

4.7 Segmentacja rozrostowa

Segmentacja rozrostu obszaru (region grow algorithm) oparta jest na porównywaniu wartości pikseli z pewną wartością referencyjną segmentu, do którego piksel jest przyłączany. Jeśli ta różnica jest mniejsza od zadanej tolerancji, piksel jest dołączany do obiektu, następuje korekcja wartości referencyjnej, a dodany piksel staje się ziarnem rozrostu, którego otoczenie badane jest dalej na możliwość dołączenia do segmentu (Rys. 39). Jest to zatem z definicji procedura rekurencyjna, jednak ze względu na dużą liczbę przetwarzanych pikseli i możliwość przepełnienia stosu, zazwyczaj jest ona zastępowana wersją z pętlą. Referencją jest zazwyczaj wartość uśredniana z wartości pikseli należących do segmentu. Sposób podziału segmentacyjnego w metodzie silnie zależy zatem od wyboru piksela początkowego, ponieważ pierwsze piksele silnie wpływają na wartość tej średniej. Optymalnym w tym sensie wyborem na ziarno jest piksel, który jest najbardziej podobny do swojego otoczenia, spośród wszystkich innych pikseli obiektu. Ze względu na to, że przy dużej liczbie pikseli w segmencie, wpływ pojedynczych pikseli na referencję jest niewielki, nowo dołączane piksele muszą mieć podobne wartości, a sam obiekt nie „rozlewa się" na obszary podobne do granic obiektu, ale niepodobne do wartości referencyjnej. Wadą algorytmu jest natomiast tworzenie drobnych obiektów na krawędziach pomiędzy dobrze rozpoznanymi obiektami, na skutek dużych różnic w wartościach pikseli.

Kryterium decydujące o przynależności piksela do segmentu można zmodyfikować, uwzględniając oprócz odległości badanego piksela od wartości referencyjnej, również jego odległość od aktualnego ziarna rozrostu (którego ten piksel jest sąsiadem). W ten sposób można wyostrzyć warunek przyłączenia piksela, odrzucając piksele, które mimo podobieństwa do wartości referencyjnej, zbyt odbiegają wartością od swoich sąsiadów.



Rys. 39. Wynik segmentacji rozrostowej dla przykładowego obrazu monochromatycznego przy założonej tolerancji równej 3. Na krawędziach podane zostały wartości referencyjne w momencie sprawdzania możliwości przyłączenia piksela do segmentu.

```
public void Seed()
{
    TSegmenter owner = Owner;
    for (int actIndex = 0; actIndex < Parent.Children.Count; actIndex++)
    {
        TSegment refSeg = Parent.Children[actIndex];
        for (int j = 0; j < refSeg.Neighbors.Count; j++)
        {
            TSegment curSeg = refSeg.Neighbors[j];
            if (curSeg.Parent == owner.Root)
            {
                double grad = curSeg.DistanceTo(Parent);
                if (grad <= owner.Tolerance)
                    curSeg.Parent = Parent;
            }
            else
                Parent.AddNeighbor(curSeg.Parent);
        }
    }
}
```

4.8 Segmentacja wododziałowo – rozrostowa

Na potrzeby projektu opracowany został algorytm hybrydowy segmentacji, który jest połączeniem segmentacji rozrostu obszaru i segmentacji wododziałowej opartej na optymalnym drzewie spinającym [65]. Algorytm łączy w sobie zalety obu metod – związanie pikseli wartością referencyjną, dzięki czemu algorytm nie „rozlewa się" na inne obiekty, oraz dołączanie krawędzi do obiektu. W pierwszej fazie każdej iteracji wyznaczany jest strumień płynący w kierunku lokalnego minimum zgodnie z

algorytmem wododziałowym OSF. Po utworzeniu strumienia i znalezieniu minimum lokalnego wykonywany jest algorytm rozrostu obszaru wokół minimum. Warto zwrócić uwagę na to, że minimum lokalne jest optymalnym położeniem dla piksela stanowiącego ziarno dla algorytmu rozrostu. Jest to bowiem piksel najbardziej podobny do swojego otoczenia. W kolejnych iteracjach strumienie docierają do już istniejących strumieni, powiększając obiekty, lub wyznaczają nowe minimum lokalne i je powiększają (łącząc przy tym pobliskie minima), zapoczątkowując nowy segment. Rozrost obszaru wokół minimum ogranicza nadsegmentację, typową dla algorytmu wododziałowego, w związku z czym nie jest konieczna kwantyzacja modułu gradientu pomiędzy pikselami. Algorytm nie tworzy również drobnych obiektów na krawędziach obszarów – są one dołączane dzięki budowie strumieni do jednego segmentu-zlewiska (Rys. 40). W przypadku obrazów rzeczywistych, o większej liczbie obiektów, algorytm hybrydowy jest zazwyczaj również szybszy od segmentacji składowych (Tabela 1).

```
public void WaterSeed()
{
    TRootSegment newRoot = new TRootSegment(this);
    for (int i = 0; i < Root.Children.Count; i++)
    {
        TSegment refSeg = Root.Children[i];
        if (refSeg.Parent != Root) continue;
        TSegment stream = new TSegment();
        stream.Parent = newRoot;
        refSeg.Parent = stream;
        refSeg = refSeg.Flow();
        if (refSeg.Parent == stream)
            refSeg.Seed();
        else
            newRoot.Children.RemoveAt(newRoot.Children.Count - 1);
    }
    Root = newRoot;
    FDest = null;
}
```

W momencie wyznaczenia minimum lokalnego, obiektem jest strumień spływający do tego minimum i posiadający wartość referencyjną będącą średnią z wartości wszystkich pikseli tego obiektu. Aby zminimalizować wpływ tych wartości na część rozrostową algorytmu, można tymczasowo usunąć piksele strumienia z obiektu, ustalając wartość referencyjną obiektu na wartość znalezionego minimum lokalnego. Po przeprowadzeniu fazy rozrostu piksele strumienia można ponownie przyłączyć do konstruowanego obiektu.

Rys. 40. Porównanie segmentacji: wododziałowej (b), rozrostowej (c) i hybrydowej (d). Założono sąsiedztwo ośmiopikselowe. Obraz oryginalny (a).

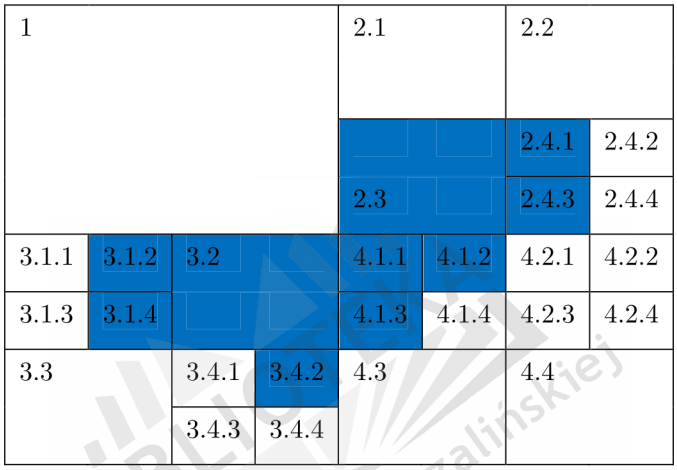
Tabela 1. Porównanie segmentacji wododziałowej, rozrostowej i hybrydowej

	wododziałowa	rozrostowa	hybrydowa
Czas obliczeń [s]	0.723	0.695	0.690
Liczba obiektów	3594	1821	880

4.9 Segmentacja Split & Merge

W metodzie tej obraz jest dzielony na ćwiartki, sprawdzane pod kątem spójności – przynależności do jednego obiektu. Jeśli obszar nie spełnia warunku spójności, jest dzielony rekurencyjnie na kolejne ćwiartki do momentu osiągnięcia jednorodnego obszaru, o spójności większej od założonego progu lub osiągnięcia minimalnego rozmiaru, np. pojedynczego piksela (Rys. 41). Po fazie podziału następuje faza łączenia spójnych obszarów w większe całości. O koszcie obliczeniowym algorytmu decyduje przede wszystkim właśnie faza łączenia (merge). Miarą spójności obszaru jest w implementacji maksymalna odległość pikseli segmentu od jego wartości referencyjnej (średniej). Algorytm tworzy nowy korzeń hierarchii do którego listy dzieci wpisywany jest dotychczasowy korzeń. Lista dzieci nowego korzenia jest przeglądana i sprawdzana jest ich spójność (jednorodność). Jeśli dziecko nie jest jednorodne, następuje jego podział na cztery części z przypisaniem pikseli do odpowiednich ćwiartek na podstawie ich położenia. Ćwiartki stają się dziećmi

korzenia hierarchii i w następnych iteracjach sprawdzana jest z kolei ich spójność i następuje ewentualny podział rekurencyjny. Segmenty jednorodne zapamiętywane są w tablicy *leaves* – są to bowiem liście budowanego drzewa czwórkowego. Na podstawie tej tablicy, dla każdego jednorodnego segmentu wyznaczany jest zbiór współrzędnych określający obrys segmentu. Jeśli punkt obrysu wyznacza piksel istniejący w obrazie, segment zawierający ten piksel zapamiętywany jest jako sąsiad przetwarzanego segmentu. Jeśli kolejne piksele obrysu są dziećmi tego samego segmentu, wpis do listy sąsiadów jest pomijany, aby nie powielać tej samej informacji. Do łączenia posłużył algorytm rozrostu obszaru. Złożoność obliczeniowa algorytmu jest rzędu *O(nlogn)*.



Rys. 41. Faza dzielenia rekurencyjnego obrazu na segmenty jednorodne

```
public void SplitAndMerge()
{
    List<TSegment> leaves = new List<TSegment>();
    TRootSegment newRoot = new TRootSegment(this);
    newRoot.Children.Add(Root);
    for ( int i = 0; i < newRoot.Children.Count; i++)
    {
        TSegment seg = newRoot.Children[i];
        if (seg.Homogeneous)
            leaves.Add(seg);
        else
        {
            TSegment[,] quarters = new TSegment[2, 2];
            for (int j = 0; j < 4; j++)
            {
                TSegment newSeg = new TSegment();
                newSeg.Parent = newRoot;
                quarters[j / 2, j % 2] = newSeg;
            }
            for (int j = 0; j < seg.Children.Count; j++)
            {
                TSegment child = seg.Children[j];
                double w = (seg.BBox.Width + 1) / 2.0;
                double h = (seg.BBox.Height + 1) / 2.0;
                double x = (child.Origin.X - seg.BBox.Left) / w;
                double y = (child.Origin.Y - seg.BBox.Top) / h;
                child.Parent = quarters[(int)y, (int)x];
```

```
            }
        }
    }
    newRoot = new TRootSegment(this);
    for (int i = 0; i < leaves.Count; i++)
    {
        TSegment seg = leaves[i];
        seg.Parent = newRoot;
        List<Point> bounds = new List<Point>();
        for (int x = seg.BBox.Left - 1; x < seg.BBox.Right + 1; x++)
            bounds.Add(new Point(x, seg.BBox.Bottom + 1));
        for (int x = seg.BBox.Left - 1; x < seg.BBox.Right + 1; x++)
            bounds.Add(new Point(x, seg.BBox.Top - 1));
        for (int y = seg.BBox.Top - 1; y < seg.BBox.Bottom + 1; y++)
            bounds.Add(new Point(seg.BBox.Right + 1, y));
        for (int y = seg.BBox.Top - 1; y < seg.BBox.Bottom + 1; y++)
            bounds.Add(new Point(seg.BBox.Left - 1, y));
        TSegment neighbor = null;
        Rectangle boundsBox = new Rectangle(0, 0, Width, Height);
        for (int n = 0; n < bounds.Count; n++)
        {
            Point p = bounds[n];
            if (!boundsBox.Contains(p)) continue;
            TSegment newNeighbor = Segments[p.X, p.Y].Parent;
            if (newNeighbor != neighbor)
            {
                seg.Neighbors.Add(newNeighbor);
                neighbor = newNeighbor;
            }
        }
    }
    Root = newRoot;
    Seed();
    FDest = null;
}
```

4.10 Redukcja małych obiektów

Po wykonaniu segmentacji jednym z opisanych algorytmów, każdy segment posiada listę segmentów podrzędnych i uchwyt do segmentu-rodzica, tworząc hierarchię segmentów w postaci drzewa. Kolejny poziom drzewa to wynik segmentacji obrazu posegmentowanego na poprzednim poziomie. Dzięki implementacji klasy segmentera na strukturze drzewiastej, każdy segment na swoim poziomie drzewa może posiadać listę segmentów z nim sąsiadujących. Jeśli w wyniku segmentacji niektóre utworzone obiekty są zbyt małe, można dołączyć je do najbardziej podobnych obiektów sąsiednich. Operacja taka została określona jako redukcja. Metoda *Reduce* klasy *TSegmenter* wymaga podania pojedynczego parametru - wielkości minimalnej obiektu określonej liczbą jego pikseli. Przeglądana jest lista dzieci dotychczasowego korzenia hierarchii, a segmenty, które mają więcej pikseli od obiektu minimalnego, wstawiane są do nowego poziomu w nowym drzewie hierarchii segmentera. Pozostałe, małe segmenty, zapamiętywane są w liście *smallSeg*. Lista ta jest następnie przeglądana, a segmenty przypisywane są do rodzica najbliższego im dużego segmentu. Jeśli wszyscy sąsiedzi segmentu są również małymi segmentami, segment jest zapamiętywany w

nowej tablicy smallSeg, zastępującej starą. Tablica smallSeg jest przeglądana tak długo, póki segmenty znajdują nowych rodziców.

```
public void Reduce(int maxSize)
{
    TRootSegment oldRoot = Root;
    Root = new TRootSegment(this);
    List<TSegment> smallSeg = new List<TSegment>();
    for (int i = 0; i < oldRoot.Children.Count; i++)
    {
        TSegment refSeg = oldRoot.Children[i];
        if (refSeg.PixelCount > maxSize)
        {
            TSegment newParent = new TSegment();
            newParent.Parent = Root;
            refSeg.Parent = newParent;
        }
        else
            smallSeg.Add(refSeg);
    }
    do
    {
        List<TSegment> oldSmallSeg = smallSeg;
        smallSeg = new List<TSegment>();
        smallSeg.Capacity = oldSmallSeg.Count;
        for (int i = 0; i < oldSmallSeg.Count; i++)
        {
            TSegment refSeg = oldSmallSeg[i];
            double minGrad = Double.MaxValue;
            int idx = -1;
            for (int j = 0; j < refSeg.Neighbors.Count; j++)
            {
                TSegment curSeg = refSeg.Neighbors[j];
                if (curSeg.Parent != oldRoot)
                {
                    double grad = refSeg.DistanceTo(curSeg);
                    if (grad < minGrad)
                    {
                        idx = j;
                        minGrad = grad;
                    }
                }
            }
            if (idx < 0)
                smallSeg.Add(refSeg);
            else
                refSeg.Parent = refSeg.Neighbors[idx].Parent;
        }
        if (smallSeg.Count == oldSmallSeg.Count) break;
    } while (true);
    FDest = null;
}
```

4.11 Sprzężenie zwrotne

Segmentację można poprawiać, w sensie ograniczenia liczby obiektów, poprzez wprowadzenie ujemnej pętli sprzężenia zwrotnego. Do obrazu wejściowego można dodać część obrazu wyjściowego, tzn. posegmentowanego, przyjmując za kolor

każdego segmentu średni kolor ze wszystkich jego pikseli. Taki obraz można powtórnie poddać segmentacji. Procedurę tę można iteracyjnie powtarzać. Efekt sprzężenia jest podobny do redukcji, ale bardziej kosztowny obliczeniowo.

W przypadku sekwencji obrazów wideo, można wykorzystać sprzężenie do poprawy stabilności segmentacji i powiązać obraz źródłowy nie tylko z wynikiem segmentacji aktualnego kadru, ale również kadru poprzedniego (lub poprzednich)

4.12 Porównanie jakości segmentacji

Algorytmy segmentacji zostały porównane z wykorzystaniem projektu „The Berkeley Segmentation Dataset and Benchmark" w wersji BSDS500 zawierającym bazę 200 naturalnych obrazów, z których każdy poddany został ręcznej segmentacji przez średnio 5 osób. Uśrednienie wyników tych segmentacji stanowi referencję prawdziwego podziału (ground truth) dla każdego z obrazów. Testy obejmują benchmark algorytmów jako detektorów konturów oraz regionów obiektów. Do oceny segmentacji regionów wykorzystane zostały popularne wskaźniki – Probabilistic Rand Index (PRI), Variation of Information (VI) i Ground Truth Covering (C). Wyniki przedstawione są w Tabela 2 i na Rys. 42. Ocena detekcji krawędzi przedstawiona została jako wykres wartości recall/precision dla różnych wartości progu detekcji krawędzi (Rys. 43).

Rand Index *RI* [66] jest miarą podobieństwa obrazu segmentacyjnego X do obrazu referencyjnego Y, określoną jako liczba par pikseli (p_i, p_j) przypisanych do tych samych segmentów w obu obrazach łącznie z liczbą par pikseli przypisanych do różnych segmentów w obu obrazach, w stosunku do liczby wszystkich par:

$$RI = \frac{a+b}{\binom{n}{2}} \tag{4.41}$$

$$a = |(p_i, p_j) \in X_k, Y_l|$$

$$b = |(p_i, p_j) \notin X_k, Y_l|$$

Wartość *RI* mieści się w znormalizowanym zakresie <0,1>. Im wartość wskaźnika większa, tym segmentacja jest poprawniejsza. Probabilistic Rand Index jest uśrednionym wskaźnikiem Rand Index dla wszystkich segmentacji referencyjnych obrazu.

Variation of Information *VI* [67] jest miarą odległości segmentacji od wzorca, definiowaną poprzez entropię obu segmentacji *H(X), H(Y),* oraz wspólną informację (transinformację) *MI* (mutual information)

$$VI = H(X) + H(Y) - 2MI(X,Y) =$$

$$= -\sum_x p(x)\log p(x) - \sum_y p(y)\log p(y) - 2\sum_{x,y} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} \tag{4.42}$$

Taką definicję można uprościć, określając VI na podstawie entropii warunkowych obu segmentacji względem siebie, bowiem:

$$VI = H(X) + H(Y) - 2MI(X,Y) = H(X|Y) + H(Y|X) =$$

$$= \sum_{x,y} p(x,y)\log\frac{p(x)}{p(x,y)} + \sum_{x,y} p(x,y)\log\frac{p(y)}{p(x,y)}$$

$$VI = \sum_{x,y} p(x,y)\log\frac{p(x)p(y)}{p(x,y)^2}, \tag{4.43}$$

gdzie:

$$n = \sum_i |X_i| = \sum_j |Y_j|, \qquad p(x) = \frac{|X_i|}{n}, \qquad p(y) = \frac{|Y_j|}{n}, \qquad p(x,y) = \frac{|X_i \cap Y_j|}{n}$$

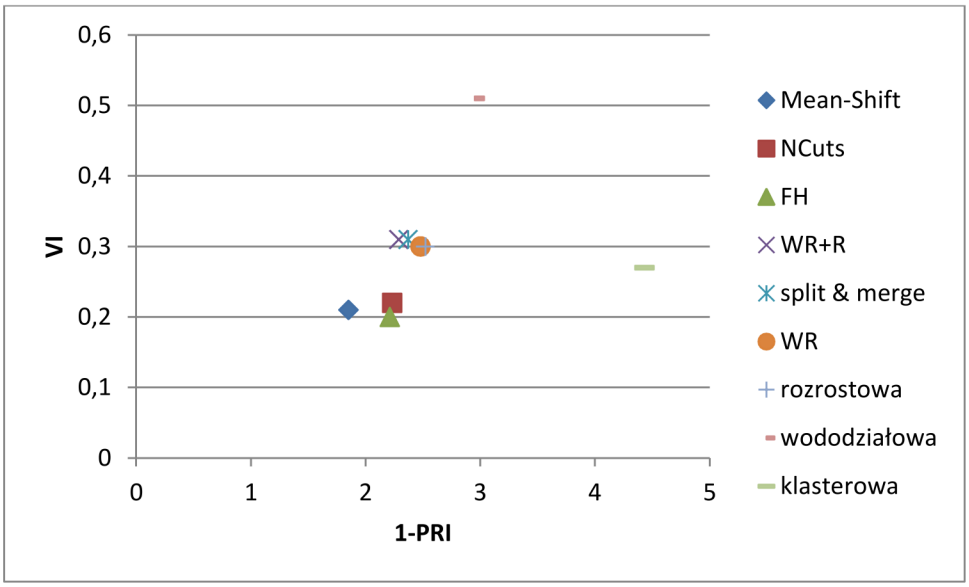
Ponieważ VI jest miarą odległości, mniejsze wartości określają lepszą segmentację.

Ground Truth Covering C jest średnią ważoną z liczby pikseli w segmentach, określającą stopień pokrywania się segmentów w obrazie segmentacyjnym X i referencyjnym Y:

$$C = \frac{\sum_k |X_k| \max_l \frac{|X_k \cap Y_l|}{|X_k \cup Y_l|}}{\sum_k |X_k|} \tag{4.44}$$

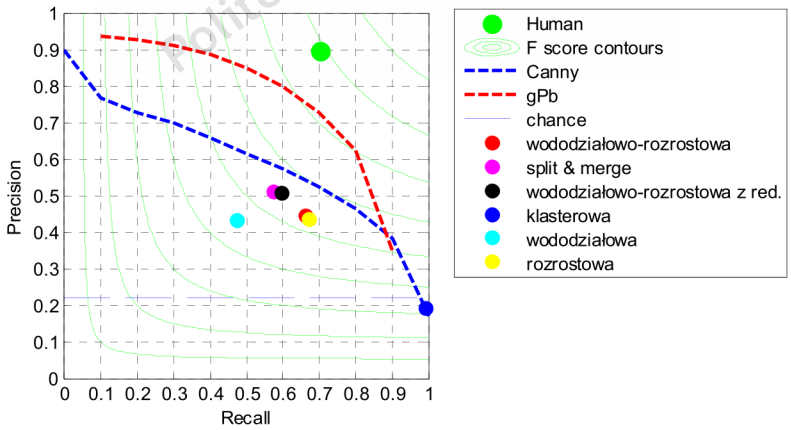
Tabela 2. Porównanie jakości segmentacji współczynnikami PRI, VI i GT Covering

Metoda\Miara jakości	PRI	VI	C
Mean-Shift	0,79	**1,85**	**0,54**
NCuts	0,78	2,23	0,45
FH (Felzenszwalb & Huttenlocher)	**0,80**	2,21	0,52
Metody zaimplementowane			
Wododziałowo-rozrostowa z redukcją WR+R	0,69	2,29	0,48
Split & merge	0,69	2,37	0,48
Wododziałowo-rozrostowa WR	0,7	2,48	0,47
Rozrostowa	0,7	2,52	0,47
Wododziałowa	0,49	2,95	0,35
Klasterowa	0,73	4,43	0,23



Rys. 42. Jakość segmentacji na wykresie VI - PRI.
Segmentacje bliższe środka układu współrzędnych określają lepsze segmentacje

Jakość detekcji krawędzi przedstawiona jest na wykresie wartości precision i recall. Precision stanowi prawdopodobieństwo tego, że piksel rozpoznany przez segmentację jako należący do krawędzi, jest w istocie jej częścią. Określa on poziom szumu na wyjściu detektora. Parametr recall jest prawdopodobieństwem rozpoznania piksela należącego do prawdziwej krawędzi, a więc jest miarą podobieństwa do segmentacji referencyjnej. Pomiary mogą być wykonywane dla różnych poziomów progu rozpoznania pikseli krawędziowych, jeśli obrazy z detektora nie są binarne i są wówczas przedstawione w postaci krzywych (Rys. 43).



Rys. 43. Jakość segmentacji na wykresie Precision - Recall

Detektory konturów przeważnie nie umożliwiają jeszcze segmentacji, ze względu na nieciągłość krawędzi. Tym niemniej dobra detekcja krawędzi ułatwia budowanie efektywnych algorytmów segmentacji.

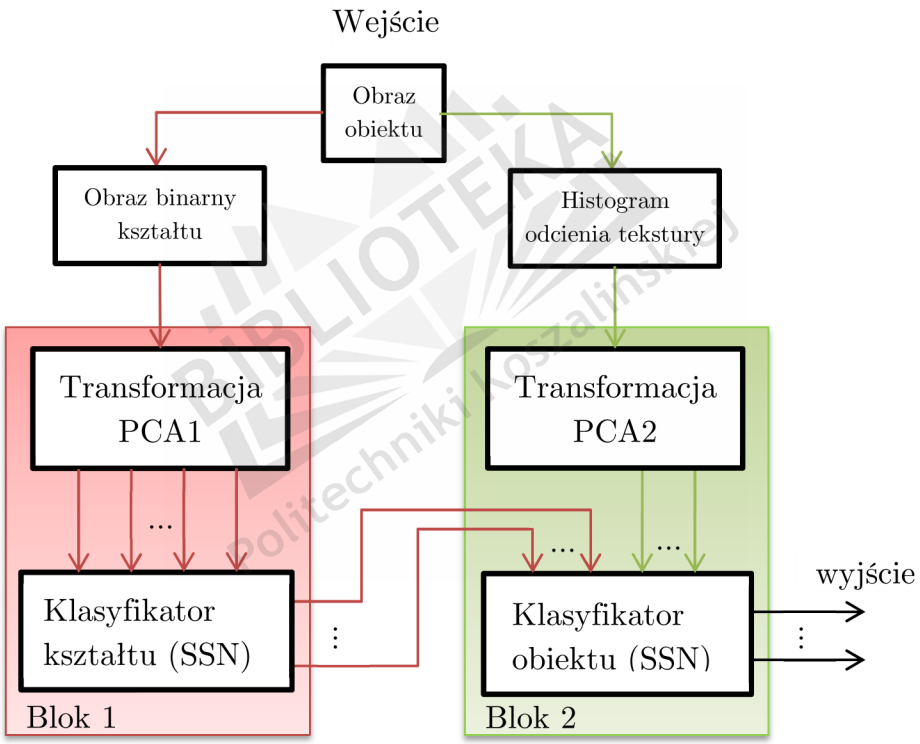
Badania wskazują, że bardziej złożone obliczeniowo algorytmy graph-cuts lub mean-shift mogą być lepsze jakościowo. Czynnikiem decydującym o wyborze analizowanych algorytmów była jednak ich szybkość. Wykorzystane algorytmy cechują się nie tylko dobrymi parametrami PRI, VI, ale również są na tyle szybkie, by umożliwić pracę w czasie rzeczywistym, przy niewielkich wymaganiach sprzętowych.

4.13 Podsumowanie

Przedstawione zostały podstawy teoretyczne metody czynników głównych PCA oraz diagonalizacji macierzy i faktoryzacji SVD. Transformacja PCA umożliwia uniezależnienie (ortogonalizację) i redukcję liczby parametrów opisujących przetwarzane dane. Metoda rozpoznawania kształtów przez SSN wykorzystuje zmodyfikowany algorytm jednostronnego przekształcenia Jacobiego do dekompozycji SVD. Metodę czynników głównych wykorzystano do ortogonalizacji i redukcji danych w procesie uczenia sieci neuronowej rozpoznającej kształty. Porównano wyniki segmentacji obrazów z projektu Uniwersytetu Berkeley BSDS500 z segmentacją tych samych obrazów z wykorzystaniem proponowanych algorytmów: kmeans, split & merge, wododziałowym, rozrostowym i wododziałowo-rozrostowym. Najlepszy rezultat uzyskano dla opracowanej segmentacji hybrydowej, będącej połączeniem algorytmu wododziałowego i rozrostowego i wykorzystującej dodatkowo redukcję liczby segmentów przez dołączenie zbyt małych segmentów do najbardziej podobnych sąsiadów.

5. Moduł rozpoznawania obiektów za pomocą sieci neuronowej

Do wspomagania rozpoznawania obiektów sceny, opartego na transformacji obraz – dźwięk, zastosowano rozpoznanie obiektu na podstawie kontekstu, kształtu i tekstury wykorzystujące sztuczną sieć neuronową. Kontekst w tym przypadku może oznaczać m.in. mieszkanie, miejsce pracy, ulicę itp., i wiąże się z ograniczonym zasobem konkretnych obiektów podlegających rozpoznaniu. Ze względu na to ograniczenie, każdy kontekst wymaga odrębnego modułu rozpoznania, pokazanego na Rys. 44.



Rys. 44. Moduł rozpoznawania obiektów

Moduł rozpoznania składa się z dwóch bloków PCA-SSN. Pierwszy blok dokonuje rozpoznania kształtu, a drugi, wykorzystując informację o kształcie oraz teksturze dokonuje właściwego rozpoznania obiektów sceny.

Danymi wejściowymi do analizy czynnikowej dla bloku pierwszego są binarne obrazy podstawowych kształtów geometrycznych (Rys. 45). Po wyznaczeniu ładunków czynników głównych i odrzuceniu komponentów o małych ładunkach, sieć klasyfikatora kształtów uczona jest rozpoznawania tych podstawowych kształtów.

Rys. 45. Przykładowy zestaw uczący klasyfikatora kształtów

Danymi dla analizy czynnikowej bloku drugiego są histogramy obrazów uczących klasyfikator obiektów z zestawu odpowiadającego konkretnemu kontekstowi (Rys. 46). Blok PCA ortogonalizuje dane wyjściowe i dzięki temu większość składowych, o małych współczynnikach istotności, można pominąć. Zmniejsza to liczbę neuronów SSN i przyspiesza proces uczenia.

Rys. 46. Przykładowy fragment zestawu uczącego klasyfikatora obiektów reprezentujący jeden obiekt

Na wejście sieci neuronowej klasyfikatora obiektów podawana jest informacja ze wszystkich wyjść sieci klasyfikującej kształt oraz dodatkowo, wynik transformacji PCA histogramu odcienia tekstury rozpoznawanego segmentu. Dlatego dopiero po wytrenowaniu sieci klasyfikującej kształt, możliwe jest uczenie sieci rozpoznającej obiekty. Wektorami uczącymi klasyfikator obiektów są rzeczywiste obiekty związane z konkretnym kontekstem sceny, w różnych ujęciach perspektywy (Rys. 46).

Danymi wejściowymi do rozpoznawania są obrazy segmentów, przeskalowane do jednakowej rozdzielczości obrazów uczących (domyślnie 64x64 piksele) z zachowaniem proporcji prostokąta opisującego segment. Wybierając rozdzielczość obrazów uczących, określa się zatem rozdzielczość obrazów segmentów do jakiej będą one skalowane i decyduje o dokładności przetwarzanej informacji.

Pojedyncza obserwacja analizy PCA1 jest wektorem utworzonym z sekwencji kolejnych wierszy obrazu uczącego klasyfikator kształtów. Ponieważ parametrami ją opisującymi są wartości pikseli które są ze sobą porównywalne, metoda odejmuje wektor wartości średnich od każdej obserwacji i dokonuje analizy macierzy kowariancji, a nie korelacji, danych.

$X_{PCA1} =$

Wektor obrazu kształtu uczącego 1
Wektor obrazu kształtu uczącego 2
...
Wektor obrazu kształtu uczącego n

Rys. 47. Macierz obserwacji PCA1

Dane do analizy czynnikowej w bloku drugim (X_{PCA2}) są wektorami histogramów odcieni tekstury obrazów uczących klasyfikator obiektów. Metoda czynnikowa

wykorzystuje do analizy, tak jak w przypadku PCA1, macierz kowariancji tych danych.

$X_{PCA2} =$

Wektor histogramu odcieni tekstury obiektu uczącego 1
Wektor histogramu odcieni tekstury obiektu uczącego 2
...
Wektor histogramu odcieni tekstury obiektu uczącego n

Rys. 48. Macierz obserwacji PCA2

Bloki PCA dokonują analizy podanej macierzy obserwacji z wykorzystaniem dekompozycji SVD, zapamiętując wektor wartości średnich ze wszystkich obserwacji uczących i macierz ładunków czynników głównych (*Loadings*).

Wyznaczane jest również osypisko, umożliwiające odrzucenie najmniej istotnych czynników. Poziom redukcji ustawiony został tak, by sumaryczna energia czynników odrzuconych nie przekraczała 10% energii sygnału. Liczba pozostałych czynników głównych określa liczbę wejść klasyfikatora. Po transformacji PCA, sygnał reprezentowany jest przez zredukowaną liczbę wzajemnie ortogonalnych parametrów. W przypadku klasyfikatora kształtów redukcja liczby wejść sieci SSN jest znacząca – dla standardowo przyjętej rozdzielczości segmentów, wektor wejściowy zawierający 4096 wartości pikseli jest, przy założonym 10% progu redukcji, zmniejszany do liczby 7 wejść sieci SSN dla zbioru uczącego z Rys. 45. Sieć SSN poprzedzona blokiem PCA zawiera mniej neuronów w warstwie wejściowej i ukrytej, a dodatkowo informacje na wejściach są niezależne, co powoduje, że sieć jest bardziej stabilna i uczy się szybciej. Transformacja PCA, po odjęciu wektora wartości średnich, przedstawia nowy wektor obserwacji w przestrzeni czynników głównych macierzy obrazów uczących (metoda *Run*).

```
[Serializable]
public class PCA
{
    public TVector Means;
    public TMatrix Loadings;
    public double Reduction = 0.1;
    public TMatrix Learn(TMatrix src)
    {
        TMatrix X = src.Clone();
        Means = new TVector(X.RowsCount);
        for (int m = 0; m < X.ColsCount; m++)
            Means += X.Cols[m];
        Means *= 1.0 / X.ColsCount;
        for (int m = 0; m < X.ColsCount; m++)
            X.Cols[m] -= Means;
        X = X.Transpose();
        TMatrix U = null, S = null, V = null;
        X.SVD(ref U, ref S, ref V);
        TVector energy = new TVector(S.RowsCount);
        double cum = 0;
        for (int i = 0; i < S.RowsCount; i++)
```

```
        {
            cum += S[i, i];
            energy[i] = cum;
        }
        int loadCount;
        for (loadCount = 1; loadCount <= S.RowsCount; loadCount++)
            if (energy[loadCount - 1] / energy[S.RowsCount - 1] > 1 - Reduction)
                break;
        Loadings = new TMatrix(V.RowsCount, loadCount);
        for (int i = 0; i < Math.Min(V.ColsCount, Loadings.ColsCount); i++)
            Loadings.Cols[i] = V.Cols[i];
        return (X * Loadings).Transpose();
    }

    public TVector Run(TVector input)
    {
        return (input - Means) * Loadings;
    }
}
```

Zmiana kontekstu lub zbioru wektorów uczących klasyfikator wymaga wyznaczenia nowych ładunków czynników głównych. Wpływa to na liczbę redukowanych czynników głównych, a tym samym na liczbę wejść sieci neuronowej, która musi zostać przebudowana.

Każdy z klasyfikatorów ma skojarzony katalog *DataPath*, w którym znajdują się obrazy treningowe do uczenia sieci, jak również plik z danymi nauczonej sieci. Zapis i odczyt danych klasyfikatorów możliwy jest dzięki serializacji bazowej klasy *TClassifier*. Zbiór uczący zawiera kilka obrazów odpowiadających temu samemu obiektowi. Pliki różnią się wówczas drugim rozszerzeniem, określającym kolejny numer ujęcia tego samego obiektu. Nazwa pliku bez rozszerzeń określa skojarzoną z obrazem nazwę tego obiektu. Liczba rzeczywistych obiektów określa rozmiar liczby wyjść klasyfikatora obiektów.

O przynależności piksela do kształtu decyduje wartość w kanale alfa - piksele tła są przeźroczyste. Dzięki temu można odczytywać informację o kształcie obiektów również na podstawie kolorowych obrazów segmentów, co jest wykorzystane podczas uczenia klasyfikatora obiektów. Liczba poziomów odcieni histogramu, będących parametrami obserwacji bloku PCA2 klasyfikatora obiektów została określona jako pierwiastek z liczby wszystkich pikseli w obrazie segmentu. Dane wyjściowe PCA2 są normalizowane liczbą tych poziomów.

```
public TVector TClassifier.ReadShape(Pixelmap pixmap)
{
    TVector x = new TVector(pixmap.Pixels.Length);
    for (int n = 0; n < pixmap.Height; n++)
        for (int m = 0; m < pixmap.Width; m++)
        {
            int j = n * pixmap.Width + m;
            x[j] = pixmap[n, m].A > 0 ? -0.5 : 0.5;
        }
    return x;
}
```

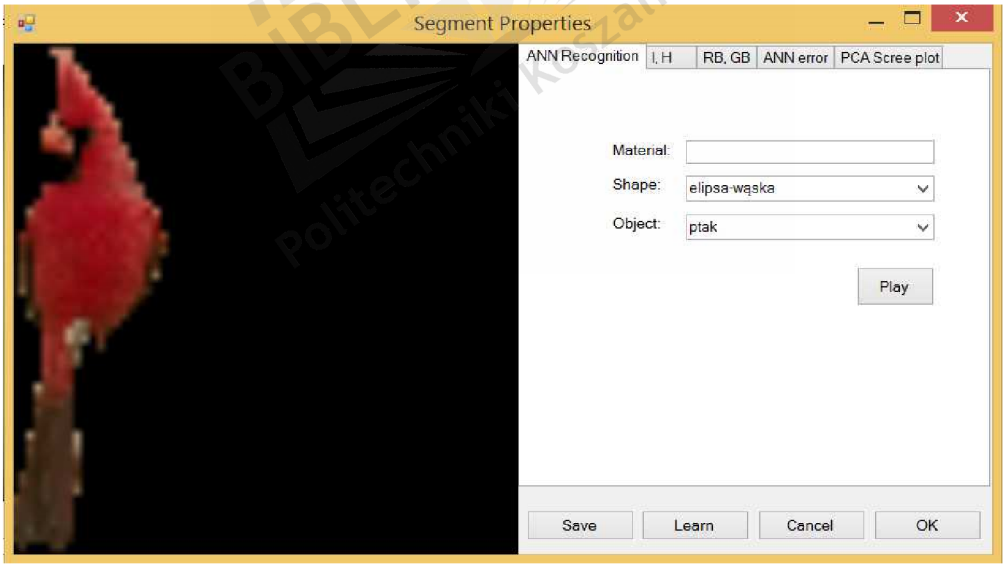
```
public TVector TClassifier.ReadHistogram(Pixelmap pixmap)
{
    TVector x = new TVector((int)Math.Sqrt(pixmap.Pixels.Length));
    double step = 1.0 / x.Count;
    for (int n = 0; n < pixmap.Height; n++)
        for (int m = 0; m < pixmap.Width; m++)
        {
            Color color = pixmap[n, m];
            if (color.A > 0)
            {
                int hue = (int)Math.Round((x.Count - 1) * color.GetHue() / 360);
                x[hue] += step;
            }
        }
    return x;
}

public List<string> Names;
public TMatrix Shapes;
public TMatrix Histograms;
public PCA PCA = new PCA();
protected virtual TMatrix TClassifier.ReadData()
{
    List<string> files = new List<string>(Directory.GetFiles(DataPath, "*.png"));
    Names = new List<string>();
    Bitmap bmp = new Bitmap(files[0]);
    int dataLength = bmp.Width * bmp.Height;
    Shapes = new TMatrix(dataLength, files.Count);
    Histograms = new TMatrix((int)Math.Sqrt(dataLength), files.Count);
    for (int i = 0; i < files.Count; i++)
    {
        string name = Path.GetFileNameWithoutExtension(files[i]);
        name = Path.GetFileNameWithoutExtension(name);
        Names.Add(name);
        Pixelmap pixmap = new Pixelmap(new Bitmap(files[i]));
        Shapes.Cols[i] = ReadShape(pixmap);
        Histograms.Cols[i] = ReadHistogram(pixmap);
    }
    return PCA.Learn(Shapes);
}
```

Implementacja sieci SSN korzysta z otwartego kodu biblioteki NeuronDotNet na licencji GPL. Wykorzystane sieci są jednokierunkowe, trójwarstwowe z jedną warstwą ukrytą o liczbie neuronów będącą średnią geometryczną liczby neuronów z warstwy wejściowej i wyjściowej. Do nauczania nadzorowanego sieci SSN wykorzystany został algorytm ze wsteczną propagacją błędu i tangensem hiperbolicznym w charakterze bipolarnej funkcji aktywacji [68].

```
public BackpropagationNetwork Cognition;
public int Cycles = 5000;
BackpropagationNetwork TClassifier.CreateNetwork()
{
    TMatrix TrainingData = ReadData();
    List<int> IDs = new List<int>();
    for (int i = 0; i < Names.Count; i++)
        IDs.Add(Names.IndexOf(Names[i]));
    int InputSize = TrainingData.RowsCount;
    int OutputSize = IDs.Count;
    TrainingSet trainingSet = new TrainingSet(InputSize, OutputSize);
```

```
    double[] outputObject = new double[OutputSize];
    for (int i = 0; i < TrainingData.ColsCount; i++)
    {
        for (int x = 0; x < OutputSize; x++)
            outputObject[x] = -0.5;
        outputObject[IDs[i]] = 0.5;
        trainingSet.Add(new TrainingSample(TrainingData.Cols[i].Items, outputObject));
    }
    LinearLayer inputLayer = new LinearLayer(InputSize);
    TanhLayer outputLayer = new TanhLayer(OutputSize);
    TanhLayer hiddenLayer = new TanhLayer((int)Math.Sqrt(InputSize * OutputSize));
    new BackpropagationConnector(inputLayer, hiddenLayer);
    new BackpropagationConnector(hiddenLayer, outputLayer);
    Cognition = new BackpropagationNetwork(inputLayer, outputLayer);
    Cognition.Learn(trainingSet, Cycles);
    return Cognition;
}
```

Klasyfikator obiektów uczony jest na podstawie obrazów umieszczonych we wspólnym podkatalogu *Objects*. Musi on zawierać początkowo przynajmniej jeden pusty obraz, określający rozmiary danych wejściowych i będący pierwszą obserwacją dla analizy PCA. Kolejne obrazy uczące mogą zostać pozyskane z segmentacji obrazu kamery, w oknie właściwości segmentu (Rys. 49). Są one wówczas automatycznie skalowane, ustawiana jest przeźroczystość tła i zapisywane są w katalogu klasyfikatora obiektów. Nowe obrazy uczące są automatycznie uwzględniane przy ponownym uczeniu klasyfikatora.

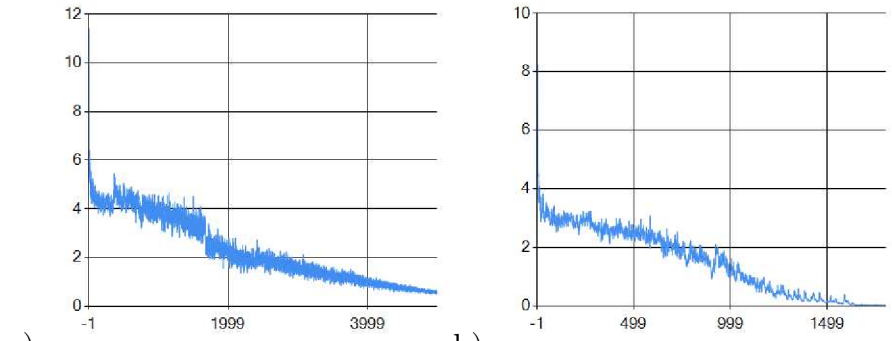


Rys. 49. Okno właściwości segmentu

W oknie tym można również przeprowadzić analizę kolorów pikseli należących do segmentu, a po nauczeniu klasyfikatora obiektów, obejrzeć historię zmian błędu średniokwadratowego MSE sieci w czasie uczenia oraz wykres osypiska bloku PCA histogramu tekstury.

Obiekty, dla których rozpoznanie jest pewne, tzn. błąd MSE nie przekracza założonej tolerancji, nie są poddawane dalszemu przetwarzaniu podczas konwersji na dźwięk

(przy włączonym module rozpoznawania przy użyciu SSN). Ich nazwy są odtwarzane przez syntezator mowy. Do poprawnego działania syntezatora w systemie Windows wymagana jest instalacja pakietu językowego i głosu Text-to-Speech (np. Paulina TTS dla języka polskiego) firmy Microsoft.



a) b)

Rys. 50. Zmiany błędu MSE sieci dla dwóch przykładowych procesów uczenia

5.1 Podsumowanie

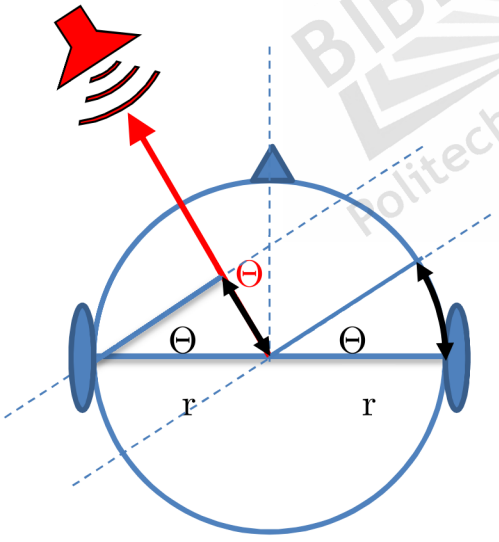
Moduł rozpoznawania obiektów przy użyciu sieci neuronowej zbudowany został z dwóch sieci neuronowych uczonych metodą wstecznej propagacji błędu i poprzedzonych blokami PCA. Pierwsza sieć jest klasyfikatorem kształtów i przeznaczona jest do rozpoznawania podstawowych kształtów geometrycznych. Sygnały wyjściowe tego klasyfikatora przekazywane są na wejście klasyfikatora obiektu wraz z informacją o teksturze obiektu, uzyskaną z bloku PCA tekstury. Klasyfikator obiektu dokonuje ostatecznego rozpoznania spośród skończonej grupy wzorców i w przypadku rozpoznania pewnego, nazwa obiektu przekazywana jest do syntezatora mowy. Zastosowanie analizy czynnikowej ortogonalizuje dane wejściowe, przez co możliwa jest również redukcja liczby wejść sieci SSN. Moduł rozpoznawania obiektów pełni funkcję wspomagającą i jego użycie może być kłopotliwe w przypadku częstej zmiany otoczenia (meble, urządzenia, przedmioty codziennego użytku).

6. Moduł syntezy dźwięku

Dźwięk charakteryzujący obiekt w obrazie segmentacyjnym może być pojedynczym akordem, ale liczba różnych tonów – polifonia – nie powinna być duża. Sam akord jest bowiem słabo rozpoznawalny. Dopiero w porównaniu z innymi akordami tworzy rozpoznawalną melodię (podobnie jak rozpoznawalność kolorów przez człowieka jest niewielka, ale już przy porównywaniu różnych kolorów ze sobą – znacznie większa). Sekwencja zmieniających się w czasie tonów lub akordów tworzy unikalną melodię, która może być łatwo identyfikowana i powiązana z informacją o kształcie rozpoznawanego obiektu.

6.1 Dźwięk przestrzenny

Parametry fali akustycznej zdefiniowane są jej widmem amplitudowym i fazowym. Dźwięk docierający do kanału słuchowego lewego ucha różni się od kanału ucha prawego ze względu na lokalizację źródła dźwięku oraz parametry związane z anatomią głowy i małżowin usznych, decydując o różnicy amplitud i faz pomiędzy tymi dźwiękami.

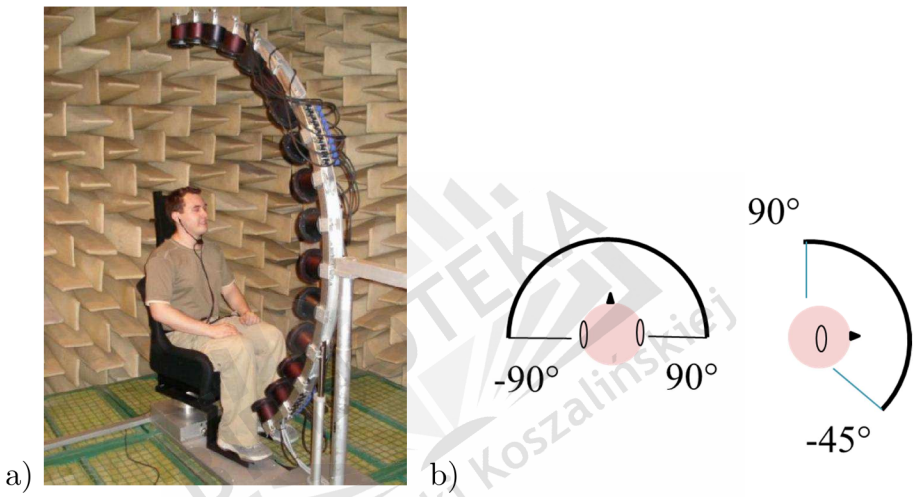


Rys. 51. Ilustracja efektu Interaural Time Difference

$$ITD = \frac{r(sin\theta + \theta)}{c} \qquad (6.1)$$

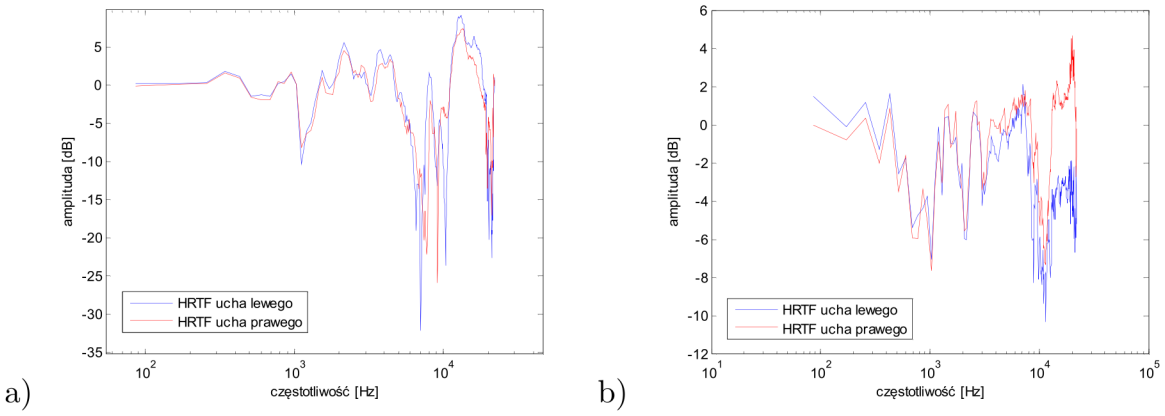
Dla wysokich częstotliwości akustycznych, długość fali jest mniejsza od rozmiarów głowy i po jednej jej stronie powstaje tzw. cień akustyczny charakteryzowany przez zmianę amplitudy pomiędzy dźwiękiem docierającym do lewego i prawego ucha - Interaural Intensity Difference (IID). Dla niższych częstotliwości, ze względu na ugięcie fali ten efekt jest dużo słabszy, ale różnicę można nadal zaobserwować dzięki

różnicy faz pomiędzy dźwiękami docierającymi do uszu – tzw. Interaural Time Difference (ITD, Rys. 51). Można to opóźnienie wyznaczyć na podstawie kąta kierunku propagacji dźwięku θ i jego prędkości c, zgodnie z równaniem (6.1). Możliwe jest również empiryczne określenie funkcji transmitancji dźwięku HRTF (Head Related Transform Function), związanej z anatomią głowy, jako widma odpowiedzi impulsowej filtru ludzkiego ucha na pobudzenie z określonej lokalizacji przestrzennej. Pomiaru dokonuje się na specjalnie przygotowanym stanowisku w komorze bezechowej, umieszczając mikrofony w lewym i prawym kanale słuchowym i umożliwiając pobudzenie impulsem z szerokiego zakresu kątów elewacji i azymutu (Rys. 52). Badania przeprowadza się przeważnie dla większej grupy osób, ze względu na związek funkcji z cechami osobniczymi.



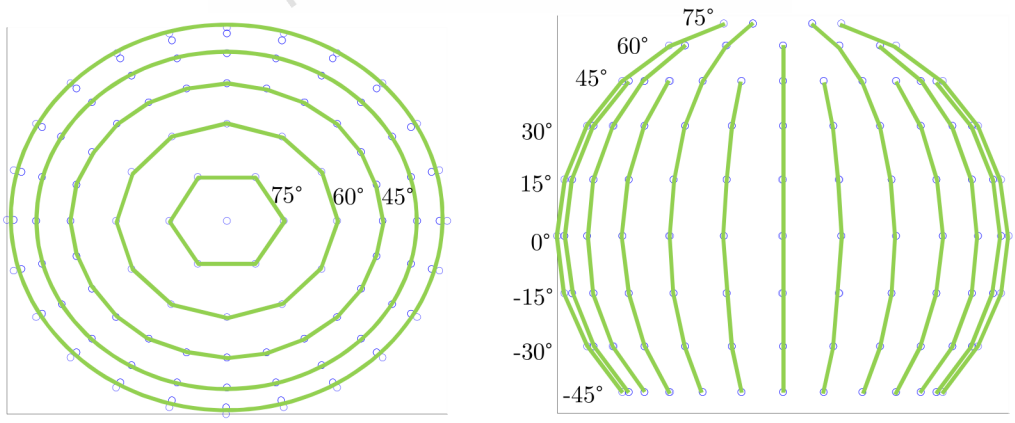
Rys. 52. Stanowisko do pomiaru funkcji HRTF w komorze bezechowej (źródło: Politechnika Łódzka) (a) i użyteczny zakres kątów azymutu i elewacji (b)

Posługując się funkcją odpowiedzi impulsowej HRIR (Head Related Impulse Response) lub jego widmem HRTF można symulować dobieganie dźwięku z określonego miejsca. Oczywiście horyzontalne położenie uszu umożliwia lepsze rozróżnianie źródeł przesuniętych w poziomie, ale m.in dzięki niesymetrycznej budowie ucha możliwe jest również postrzeganie różnicy położeń źródeł dźwięku w pionie (Rys. 53) [69].



Rys. 53. Przykładowa funkcja HRTF dla azymutu 0° i elewacji 0° (a) oraz elewacji 75° (b)

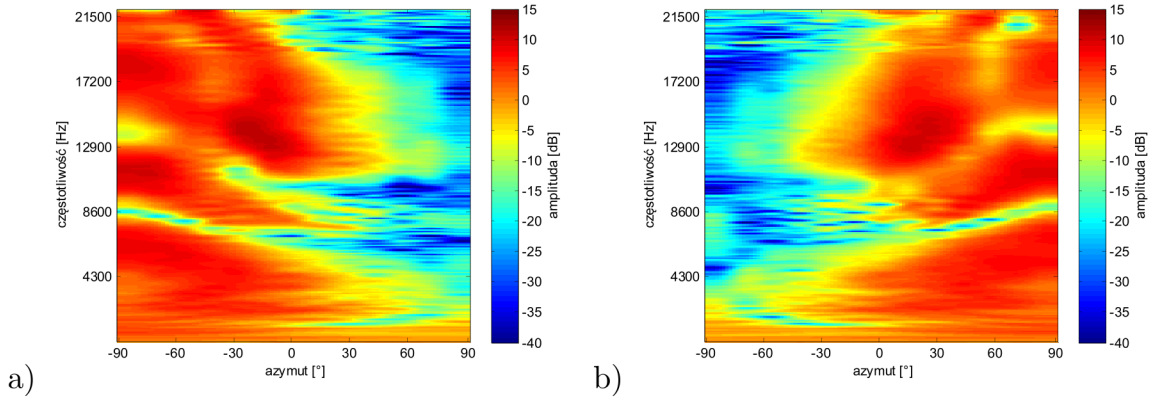
Najpopularniejszymi bazami HRTF są dane z Uniwersytetu Kalifornijskiego CIPIC [70], baza charakterystyk KEMAR uzyskana dla głowy manekina w MIT [71], oraz kompletny zbiór charakterystyk 51 osób Instytutu IRCAM w Paryżu - LISTEN [19]. Wykorzystane zostały charakterystyki oznaczone numerem 1022 z bazy LISTEN, ze względu na jakość pomiaru (m.in ciągłość funkcji ITD [72]) i największe podobieństwo do pozostałych zbiorów. Zestaw zawiera pomiary odpowiedzi impulsowych dla lewego i prawego ucha pomierzone dla kierunku dźwięku określonego przez 10 kątów elewacji od -45° do 90° (w 15° krokach). Dla każdego kąta elewacji poniżej lub równego 45° wykonano pomiary dla 24 kątów azymutu, również z krokiem 15°. Dla elewacji 60° krok zwiększono do 30°, dla 75° znów 2-krotnie, rejestrując tylko 6 charakterystyk. Zarejestrowano również impuls dla kierunku pionowego. Łącznie zarejestrowano 187 (24 * 7 + 12 + 6 + 1) stereofonicznych odpowiedzi impulsowych (Rys. 54).



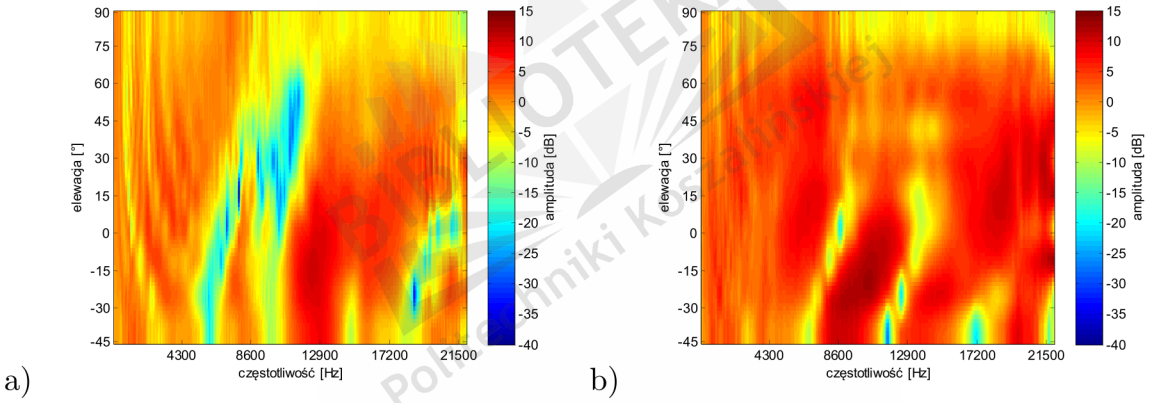
Rys. 54. Siatka lokalizacji źródeł dźwięku przy pomiarach charakterystyk HRTF bazy LISTEN

Kąt azymutu równy zeru oznacza kierunek na wprost źródła i zwiększa się w kierunku przeciwnym do wskazówek zegara (lewoskrętnie) (Rys. 52b). Dla elewacji 60° i 75° uzupełniono poprzez interpolację liniową brakujące pomiary, tak, by uzyskać

siatkę z 15° krokiem kątów azymutu. Siatka ta jest wykorzystywana do przeprowadzenia interpolacji biliniowej w celu wyznaczenia przybliżonych charakterystyk prawego i lewego ucha dla dowolnego położenia źródła dźwięku (Rys. 55, Rys. 56).



Rys. 55. Widmo amplitudowe HRTF (interpolacja bikubiczna) zbioru 1022 bazy LISTEN przy zmianie kąta azymutu od -90° do 90° na poziomie 0° elewacji dla ucha lewego (a) i prawego (b)



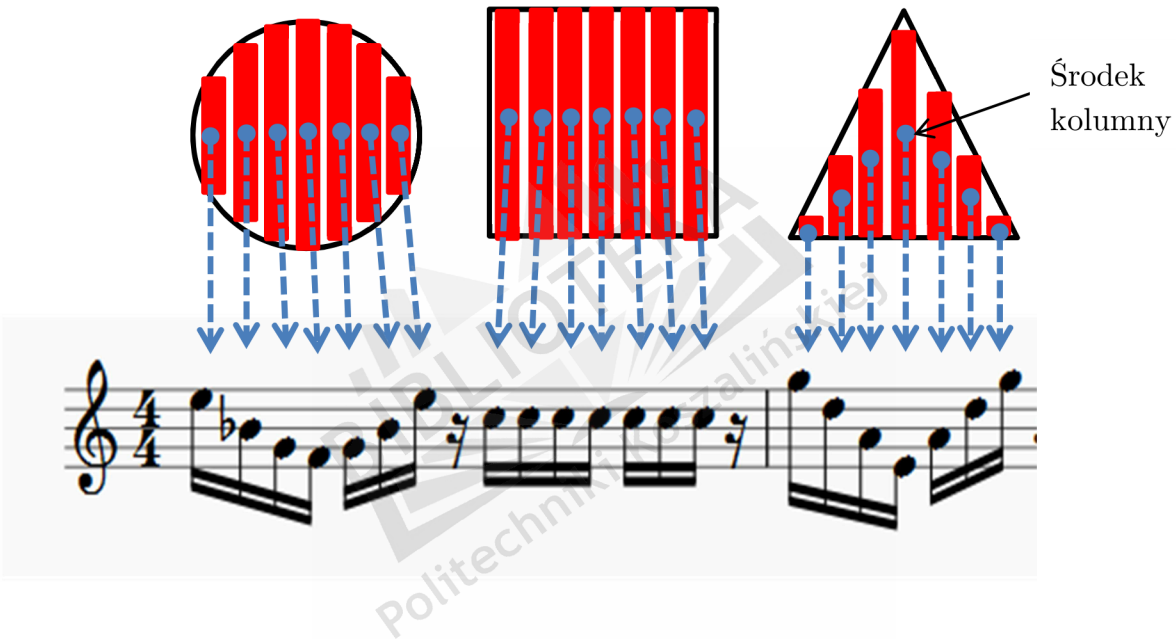
Rys. 56. Widmo amplitudowe HRTF (interpolacja bikubiczna) zbioru 1022 bazy LISTEN dla ucha lewego przy zmianie kąta elewacji od -45° do 90° dla azymutu 0° (a) i 90° (b)

Istniejące systemy konwersji obrazu albo analizują cały obraz (skanery przekazują informację o wszystkich pikselach w obrazie) bądź też obraz jest segmentowany, ale rozpoznane obiekty generują dźwięki określone parametrami związanymi jedynie z ogólną wielkością obiektów i ich położeniem. Rzadko analizie poddawany jest kształt obiektów. John Cronly-Dillon, Krishna Persaud i R.P.F. Gregory na Uniwersytecie Manchester podjęli w 1999 ciekawą próbę sonifikacji konturu obiektu, traktując go jak zapis pianoli, w którym piksel należący do konturu decyduje o wysokości granej nuty [73].

W przyjętym w dysertacji rozwiązaniu dokonano podziału obiektu na kolumny. Liczba kolumn jest parametrem przetwarzania i nie zależy od wymiarów obiektu. Szerokość kolumny wyznaczana jest z podzielenia szerokości obiektu przez liczbę kolumn. Wysokość kolumny odpowiada wysokości obiektu w miejscu jej położenia

(Rys. 57). Liczba i zakres tonów, czas odtwarzania pojedynczej kolumny oraz instrument są również parametrami przetwarzania i umożliwiają lepsze dostosowanie parametrów transformacji do indywidualnych cech użytkownika.

Wysokość tonu jest odwrotnie proporcjonalna do wysokości kolumny, którą ten dźwięk reprezentuje. Położeniem geometrycznym źródła dźwięku jest środek każdej kolumny (Rys. 57). Dźwięki odpowiadające poszczególnym kolumnom są odtwarzane sekwencyjnie od lewej do prawej. W ten sposób przekazywana jest informacja o położeniu i kształcie obiektu. O rozmiarze obiektu można sądzić na podstawie wysokości dźwięku a o kształcie decyduje zarówno wysokość jak i zmiana położenia źródła dźwięku w pionie.

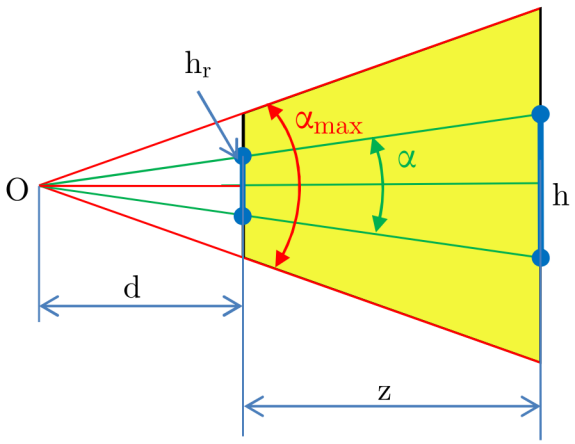


Rys. 57. Tworzenie klucza dźwiękowego obiektu - środek każdej kolumny jest źródłem dźwięku przestrzennego o częstotliwościach proporcjonalnych do długości kolumny

6.2 Ocena odległości obserwatora do obiektu

Jednym z ważniejszych elementów orientacji przestrzennej osoby niewidomej jest możliwość oceny odległości obserwatora do obiektu. Można wykorzystać do tego celu systemy stereowizyjne, ultradźwiękowe lub laserowe pomiary odległości, itp. [74]. Proponowane w ramach niniejszej dysertacji rozwiązanie nie dostarcza, w sposób jawny, takiej informacji, ale możliwe jest szacowanie zarówno wielkości obiektu jak i jego położenia względem obserwatora dzięki zależności częstotliwości dźwięku (wysokości rzutu obiektu) oraz zakresu zmian położenia źródeł dźwięku (szerokość rzutu obiektu) od rozmiarów obiektu i jego odległości od obserwatora. Efekt perspektywy powoduje, że wielkość rzutowanego obiektu zmienia się wraz z jego odległością od kamery (Rys. 58). Gdy stosunek rozmiarów poprzecznych obiektu do

odległości pomiędzy obserwatorem i obiektem jest niewielki, zmiana odległości wpływa w mniejszym stopniu na zmianę częstotliwości oraz położenia źródeł dźwięku, niż w sytuacji, gdy ten stosunek jest duży.



Rys. 58. Efekt perspektywy określony przez kąt widzenia w pionie

Rozmiar rzutu obiektu, dla rzutni umieszczonej w odległości d od środka rzutowania O, jest związany ze stosunkiem jego wysokości h do odległości z :

$$\frac{h_r}{d} = \frac{h}{z+d}$$

Dla $z \gg d$:

$$h_r \cong d\frac{h}{z}$$

Ponieważ indeks tonu n zależy od względnej wysokości rzutu obiektu,

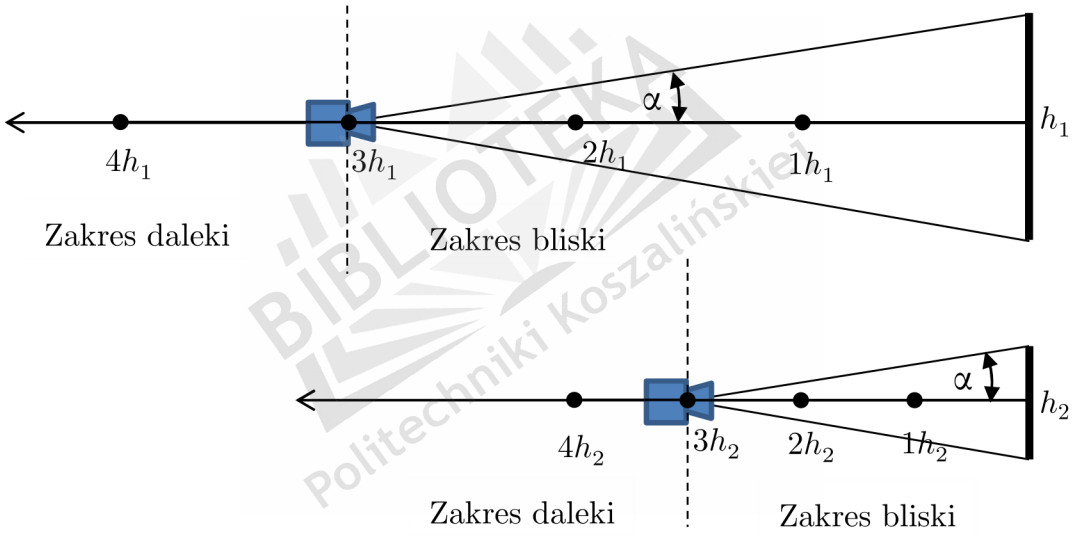
$$n = n_{max} - (n_{max} - n_{min})\frac{h_r}{h_{rmax}}$$

to szybkość zmian dźwięku jest proporcjonalna do iloczynu względnej wysokości obiektu (h/z) oraz odwrotności odległości $(1/z)$:

$$\frac{\partial n}{\partial z} = -\frac{(n_{max} - n_{min})}{h_{rmax}}\frac{\partial h_r}{\partial z} = \frac{(n_{max} - n_{min})dh}{h_{rmax}x^2} = (n_{max} - n_{min}) \cdot \frac{h}{z} \cdot \frac{1}{z}$$

Wynika stąd, że szybkość zmiany częstotliwości rośnie wraz ze wzrostem względnej wysokości obiektu i zbliżaniu się do niego. Z tego względu na podstawie zmiany częstotliwości dźwięku można sądzić o bliskości obiektu, przy czym bliskość jest pojęciem relatywnym i ma związek z jego wielkością – obiekt jest blisko, gdy bezwzględna odległość do niego jest porównywalna z jego wysokością.

W celu określenia przybliżonych granic percepcji zmian odległości do obiektu oraz jego przybliżonych wymiarów przeprowadzono eksperyment. Kamerę przemieszczano względem obiektu (czarny kwadrat na białym tle) w zakresie od 120 cm do 20 cm z krokiem równym długości boku h obserwowanego kwadratu. W trzech eksperymentach rozmiary kwadratów wynosiły 20x20, 10x10 i 5x5 cm^2. Zadaniem obserwatora było wskazanie momentu, w którym zmiana położenia kamery skutkowała zauważalną przez niego zmianą wysokości dźwięku. Niezależnie od wielkości kwadratu, obserwator wskazywał najczęściej na moment odpowiadający odległości kamery od obiektu równej w przybliżeniu $3h$. Jeśli przyjąć tę odległość za wykrywalny próg bliskości, to jest to równoznaczne z tym, że obiekt jest „blisko", jeśli stosunek jego wysokości do odległości jest większy od 1/3. Pojęcie bliskości w tym przypadku ma niestety charakter względny, ponieważ łączy bezwzględną odległość z wielkością obiektu i dodatkowo z wielkością przemieszczenia obserwatora względem obiektu.



Rys. 59. Eksperyment oceny odległości do obiektu

Ten eksperyment pokazuje jedynie pewne możliwości oceny odległości do obiektu. Trudno je uznać za w pełni satysfakcjonujące, tym bardziej, że wymienione wyżej parametry mogą zależeć od wrażliwości osobniczej obserwatora oraz przyjętej skali tonów i perspektywy sceny (parametrów kamery). Oznacza to niestety również konieczność normalizacji parametrów technicznych kamery oraz sposobu transformacji obraz – dźwięk. Każda ich zmiana może skutkować zaburzeniem wcześniej wyuczonych nawyków.

6.3 Wykorzystanie próbek instrumentów muzycznych i standardu MIDI

Dźwięk w standardzie MIDI (Musical Instruments Digital Interface) tworzony jest na podstawie otrzymywanych od aplikacji lub urządzenia zdarzeń MIDI. Są one odpowiednikiem zapisu nutowego związanego z użyciem instrumentu muzycznego, określają bowiem włączenie lub wyłączenie nuty o określonych parametrach (takich jak częstotliwość, głośność, rodzaj instrumentu muzycznego), użycie pedału, efektu, itp. Standard MIDI umożliwia granie nut poszczególnych instrumentów poprzez wysyłanie odpowiednich wiadomości do otwartego urządzenia MIDI. Instrumenty są zazwyczaj związane z kanałem MIDI. Do zmiany instrumentu (Program Change), grania lub wyciszenia nuty (Note On, Note Off) i zmiany parametrów dźwięku (Controller) służą wiadomości związane z kanałem – Channel Messages (Tabela 3). W standardowej implementacji banku MIDI, tzw. General MIDI do dyspozycji jest 16 kanałów, które pozwalają na odtwarzanie 128 różnych nut (prawie 12 oktaw) dla 128 różnych instrumentów. Częstotliwość dźwięku w oktawach zmienia się w skali logarytmicznej – ten sam ton w kolejnej oktawie ma częstotliwość dwa razy wyższą. Instrumenty podzielone są na szesnaście grup (po osiem w każdej grupie). Dodatek A wyszczególnia wszystkie dostępne instrumenty. Listę instrumentów i sposób generowania ich tonów można zmieniać poprzez użycie fontów dźwiękowych (SoundFonts). W systemie został wykorzystany software'owy syntezator MIDI, w którym próbki nut instrumentu zapisane są w postaci przekonwertowanych plików fontu dźwiękowego (z formatu sf2) w formacie sfz. Nie wszystkie dźwięki nut uzyskiwane są z oddzielnych próbek. Format umożliwia uzyskiwanie brzmienia części nut na drodze aproksymacji i filtracji dźwięków sąsiednich tonów.

Tabela 3. Wiadomości związane z kanałem MIDI

Zdarzenie	ID (4 bity)	Kanał (4 bity)	Parametr1 (1 bajt)	Parametr2 (1 bajt)
Note Off	0x8	numer kanału	numer nuty	prędkość
Note On	0x9	numer kanału	numer nuty	prędkość
Note Aftertouch	0xA	numer kanału	numer nuty	wartość aftertouch
Controller	0xB	numer kanału	numer kontrolera	wartość kontrolera
Program Change	0xC	numer kanału	numer programu	-
Channel Aftertouch	0xD	numer kanału	aftertouch	-
Pitch Bend	0xE	numer kanału	pitch LSB	pitch MSB

Standard MIDI nie przewiduje obecnie możliwości definiowania pozycji przestrzennej źródła dźwięku. System zawiera jednak programowy syntezator MIDI, w którym zaimplementowana została możliwość definiowania i interpretowania danych

lokalizacyjnych w celu stworzenia dźwięku przestrzennego, dobiegającego z określonego położenia w przestrzeni. W tym celu posłużono się współrzędnymi sferycznymi tego miejsca, określając kąt azymutu (w poziomie) i elewacji (w pionie). Po wygenerowaniu przebiegu czasowego granej nuty, musi zostać ona poddana filtracji funkcją HRTF. Są to widma filtrów uzyskiwane z odpowiedzi impulsowej zarejestrowanej w lewym i prawym uchu człowieka, przy pobudzeniu źródłem dźwięku umieszczonym w położeniu określonym przez kąty azymutu i elewacji. Dźwięki nut poddanych takiej filtracji można ze sobą zmiksować. Do przekazania informacji o położeniu źródła dźwięku zostały wykorzystane zdarzenia kontrolera MIDI: 10 (Pan – MSB, Coarse) i 42 (Pan – LSB, Fine). Kontroler Pan decyduje bowiem w standardzie MIDI o balansie pomiędzy kanałem lewym i prawym nagrania stereo. Tryb stereo zastąpiony został tym samym trybem dźwięku przestrzennego.

6.4 Filtracja dźwięku w czasie rzeczywistym

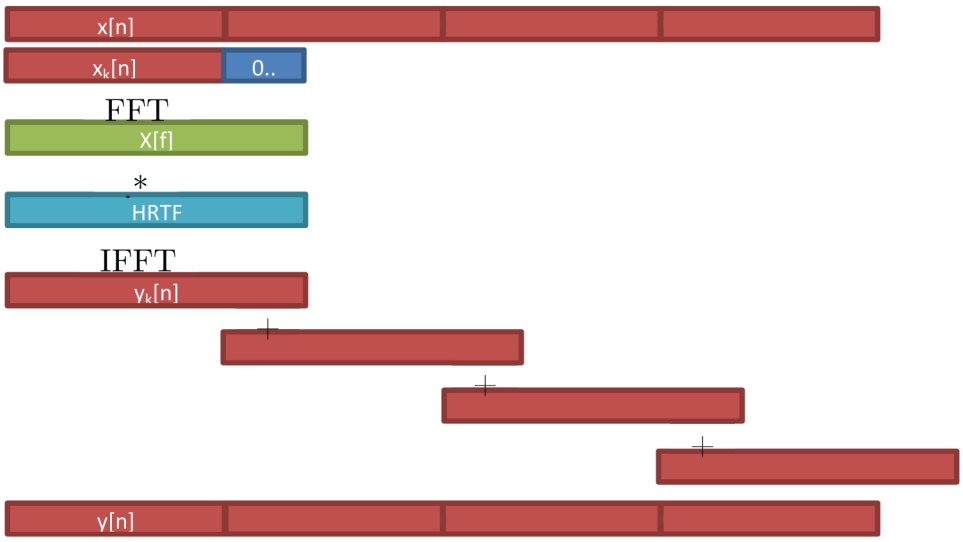
Aby dokonać szybkiej konwolucji dźwięku z odpowiedzią impulsową filtru HRIR należy wyznaczyć widma obu sygnałów i wymnożyć je element po elemencie. Posługując się FFT wyznaczamy jednak splot kołowy, ze względu na założoną periodyczność sygnału. Aby uzyskać splot liniowy musimy uzupełnić sygnały wektorami zerowymi (zero-padding) o długości M, równej długości odpowiedzi impulsowej filtru HRIR.

$$y[n] = IFFT(\ FFT(x[n])\ .*\ FFT(h[n])\ )$$

Istnieją dwie równoważne metody wyznaczania szybkiej konwolucji sygnału rejestrowanego w czasie rzeczywistym z odpowiedzią impulsową pewnego filtru o skończonej odpowiedzi impulsowej. Obie zakładają podział sygnału na odcinki o długościach L:

Metoda Overlap-Save zakłada, że odcinki zachodzą na siebie. Po konwolucji odcinka, jego przednią część się odrzuca i składa od miejsca zakończenia poprzedniego.

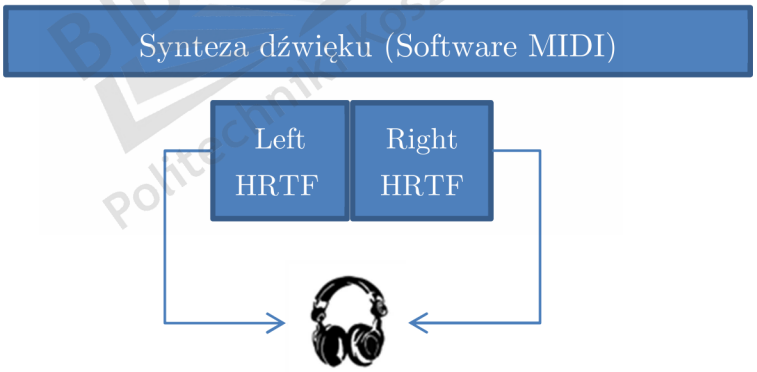
Metoda Overlap-Add zakłada, że odcinki nie zachodzą na siebie, ale są uzupełniane zerami. Po konwolucji odcinki są do siebie dodawane (Rys. 60).



Rys. 60. Przetwarzanie dźwięku w czasie rzeczywistym metodą Overlap-Add

$$y[n] = x[n] * h[n] = \left(\sum_k x_k[n - kL]\right) * h[n] = \sum_k (x_k[n - kL] * h[n]) = \sum_k y_k[n - kL]$$

W Systemie została zaimplementowana metoda Overlap-Add, ponieważ sygnał i tak składany jest poprzez dodawanie (miksowanie) odcinków związanych z aktywnymi głosami. Miksowanie zastosowane jest oddzielnie dla kanału lewego i prawego (Rys. 61).

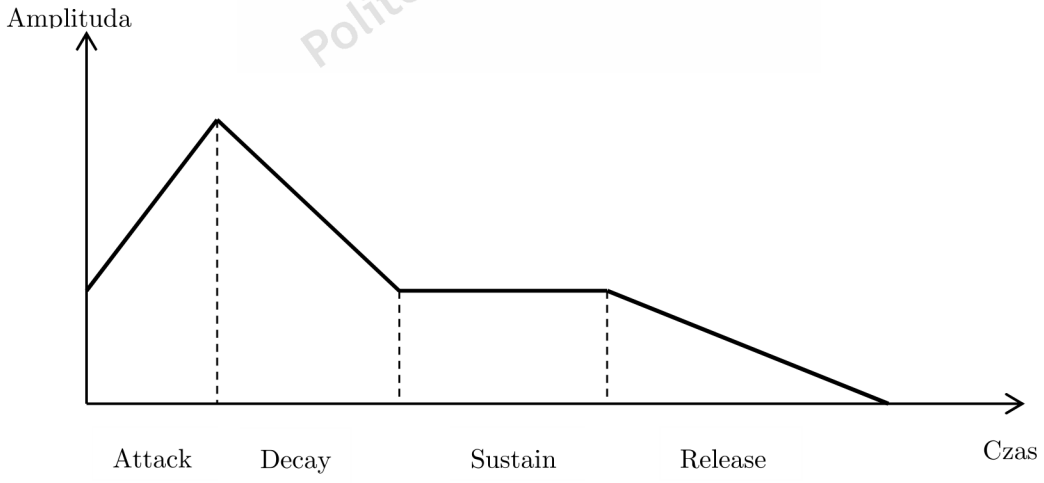


Rys. 61. Przetwarzanie dźwięku z użyciem funkcji HRTF

6.5 Synteza sygnału dźwiękowego

Do generacji dźwięku wykorzystany został software'owy syntezator MIDI utworzony w oparciu o kod projektu CSharpSyth na otwartej licencji publicznej GNU. Odtwarzanie dźwięku na podstawie obrazu segmentacyjnego wykonywane jest w oddzielnym wątku aplikacji. W ten sposób, podczas odtwarzania dźwięku może zostać wykonana znaczna część obliczeń związanych z segmentacją następnej klatki i możliwa jest realizacja systemu działającego w czasie rzeczywistym. Synchronizacja wątków zrealizowana została z użyciem prostego blokowania (mutexu).

Segment podlegający konwersji jest dzielony na stałą i jednakową dla wszystkich segmentów liczbę kolumn. Kolejne kolumny są analizowane i zapamiętywane na liście komunikatów MIDI syntezatora. Nie wszystkie instrumenty zawierają pełną gamę 128 tonów, dlatego nuty zostały ograniczone do zakresu 36 (C wielka) do 84 (C trzykreślne). Czas odtwarzania dźwięku opisującego pojedynczy obiekt określony został domyślnie na 1s, stąd, przy podziale segmentu na np. 16 kolumn, każda kolumna zamieniana jest na nutę szesnastkową. Komunikaty MIDI są przetwarzane kolejno i blokowo, umożliwiając miksowanie wielu głosów (polifonię), jak również zastosowanie szybkiej konwolucji z transmitancją HRTF metodą overlap-add w celu realizacji efektu przestrzennej lokalizacji źródła dźwięku. Przykładowo podczas analizy komunikatu MIDI NoteOn (włączenia nuty) z banku głosów pobierany jest wolny głos i ustawiane są jego parametry związane z graną nutą – kanał stereo, numer nuty (częstotliwość), prędkość (głośność), instrument oraz kąty azymutu i elewacji określające położenie przestrzenne źródła. Głos ten jest dodawany do listy aktywnych głosów. Maksymalna liczba głosów jest stała i związana z wybranym stopniem polifonii. W przypadku, gdy w banku brakuje wolnych głosów, kilka głosów aktywnych najdłużej jest zatrzymywanych (technika stealth voice). Podczas przetwarzania kolejnych komunikatów MIDI przeglądana jest lista aktywnych głosów i dla każdego z nich wywoływana jest metoda, która sumuje próbki poddane interpolacji i filtracji. Obwiednia sygnału jest aproksymowana zgodnie z odcinkami liniowym przebiegiem funkcji ADSR zawdzięczającej swoją nazwę kolejnym fazom przebiegu: Attack, Decay, Sustain i Release (Rys. 62). Implementacja funkcji jest podobna jak w przypadku funkcji zmian komponentów RGB przy budowie palety barw.



Rys. 62. Obwiednia ADSR kształtująca dźwięk instrumentów muzycznych

Każdy z głosów może znajdować się w jednym z trzech stanów: *playing*, *stopping* lub *stopped*. W momencie wyłączenia głosu (nuty), przechodzi on ze stanu *playing* do stanu *stopping*, ustawiając fazę Release obwiedni ADSR. Po wyciszeniu, głos

przechodzi do stanu *stopped* i jest dodawany do listy wolnych głosów. W Systemie wykorzystano do modulacji amplitudy tylko fazę podtrzymania (Sustain). Zakończenie nuty powoduje natychmiastowe zatrzymanie głosu. W ten sposób możliwe jest odtwarzanie kolejnych nut bez wprowadzania zakłóceń związanych z przesunięciem fazowym odtwarzanych kolumn i zaznaczaniem początku granych nut. Przy dużej liczbie kolumn powoduje ono bowiem modulację dźwięku i występowanie niepożądanych harmonicznych w przekazie. Zakres tonów instrumentów MIDI podzielony jest na podzakresy określane jako regiony i opisany w plikach fontu dźwiękowego sfz. Z każdym regionem związany jest plik próbek tonu, na podstawie którego możliwe jest wygenerowanie zakresu tonów niższych, poprzez nadpróbkowanie (upsampling) danymi uzyskanymi na drodze interpolacji. Dla takich tonów czas odtwarzania pętli fazy podtrzymania dźwięku jest odpowiednio dłuższy. Korekta fazowa uwzględnia więc nie tylko numery próbek startowych i końcowych pętli, ale również częstotliwość granej nuty, ustalając czas odtwarzania jednej kolumny na całkowitą wielokrotność czasu pętli (Rys. 63).

Rys. 63. Odtwarzanie kolumn: z obwiednią ADSR a) bez obwiedni i z korekcją fazy końcowej i początkowej nut b)

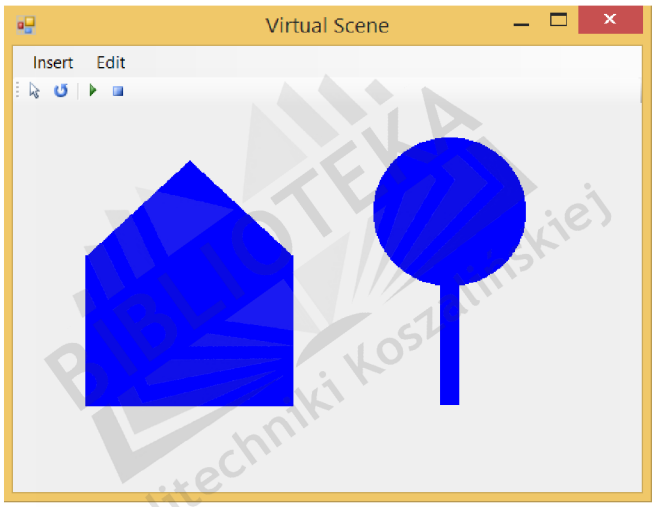
6.6 Podsumowanie

Treścią rozdziału jest przedstawienie algorytmu zamiany obrazu segmentu na postać dźwięku przestrzennego. Kształt segmentu jest reprezentowany w postaci kolumn, których kolejne długości decydują o częstotliwościach sekwencyjnie granych nut instrumentu MIDI, wykorzystanego do odtwarzania informacji o segmencie. Środek

każdej kolumny wyznacza kierunek dobiegania dźwięku, uwzględniony przez użycie filtracji wykorzystującej funkcję HRTF z bazy LISTEN [19]. Szybka filtracja zrealizowana została z wykorzystaniem algorytmu Overlap-Add. Ze względu na pracę Systemu w czasie rzeczywistym segmentacja obrazu i odtwarzanie syntezowanego dźwięku odbywa się równolegle w oddzielnych wątkach aplikacji.

7. Moduł sceny wirtualnej

Częścią Systemu jest moduł wirtualnej sceny, przeznaczony do nauki rozpoznawania rodzajów obiektów i ich położenia na podstawie dźwięku przestrzennego. Aktualna implementacja umożliwia budowanie prostych scen z wykorzystaniem edytora grafiki wektorowej 2D. Edytor umożliwia tworzenie podstawowych figur geometrycznych podlegających przekształceniom afinicznym. Możliwe jest również tworzenie dowolnie skomplikowanych kształtów dzięki funkcji grupowania i rozgrupowania (Rys. 64). Każda grupa jest również kształtem, który może podlegać transformacjom. Obraz utworzonej sceny wirtualnej może być przetworzony na dźwięk przestrzenny z wykorzystaniem podstawowego modułu.



Rys. 64. Przykładowe grupy kształtów

Grupowanie kształtów jest możliwe dzięki strukturze drzewiastej obiektów w scenie. Każdy kształt typu *TShape* posiada uchwyt *Parent* do kształtu rodzica i listę podkształtów *Children*, których układ współrzędnych jest zdefiniowany względem rodzica. Do narysowania kształtu wykorzystywana jest lista jego wierzchołków *Vertices*, pióro *Pen* do rysowania krawędzi i pędzel *Brush* do wypełnienia wnętrza.

```
public class TShape
{
    public List<PointF> Vertices = new List<PointF>();
    public Pen Pen = new Pen(Color.Blue, 0);
    public SolidBrush Brush = new SolidBrush(Color.Blue);
    public List<TShape> Children = new List<TShape>();
    TShape FParent;
    public TShape Parent
    {
        get { return FParent; }
        set
        {
                if (FParent != null)
```

```
                FParent.Children.Remove(this);
            FParent = value;
            if (FParent != null)
                FParent.Children.Add(this);
        }
    }
}
```

Okno edytora graficznego stanowi klasa kontrolki użytkownika, która podczas odświeżania rysuje korzeń hierarchii obiektów *Root*. Ponieważ procedura rysowania kształtu wywołuje się rekurencyjnie dla każdego z jego podkształtów, przy odświeżaniu odrysowana zostanie cała scena:

```
protected virtual void DrawShape(Graphics context) { }
public virtual void Draw(Graphics context)
{
    Matrix transform = context.Transform;
    context.MultiplyTransform(Transform);
    DrawShape(context);
    for (int i = 0; i < Children.Count; i++)
        Children[i].Draw(context);
    context.Transform = transform;
}
```

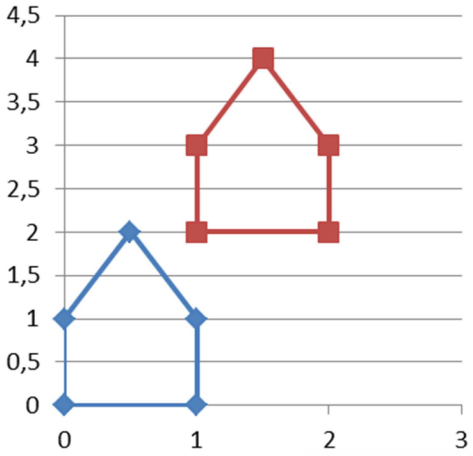
Do dyspozycji użytkownika są transformacje afiniczne na płaszczyźnie: skalowanie, pochylenie, obrót i przesunięcie. Ponieważ przesunięcie nie jest transformacją liniową, do zapisu wszystkich transformacji wykorzystywana jest macierz o jeden wymiar większa niż wynikałoby to z przestrzeni, w której znajdują się przetwarzane punkty. Jeśli punkt P' jest obrazem wierzchołka we współrzędnych jednorodnych P(x, y, 1) obiektu, względem każdego z przekształceń i przy założeniu wierszowego zapisu macierzowego, jego współrzędne można wyznaczyć przez mnożenie z następującą macierzą transformacji:

1. Przesunięcie o wektor [tx, ty]:

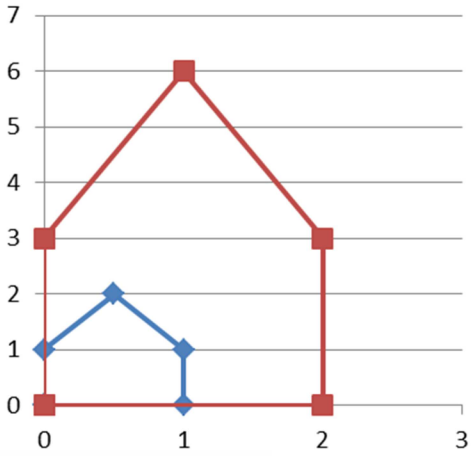
$$P' = P * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} = P * \boldsymbol{T}$$

2. Skalowanie 2D [sx, sy]:

$$P' = P * \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = P * \boldsymbol{S}$$



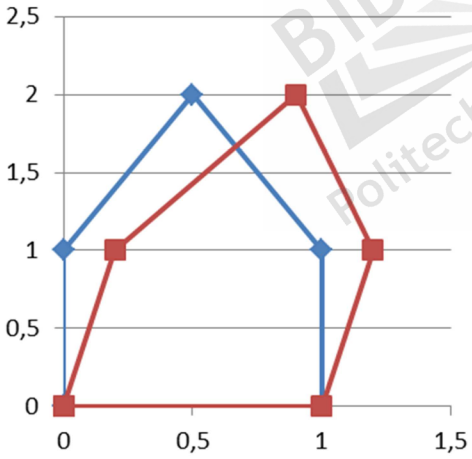
Rys. 65. Przesunięcie o wektor [tx, ty]



Rys. 66. Skalowanie 2D [sx, sy]

3. Pochylenie z parametrem *h:*

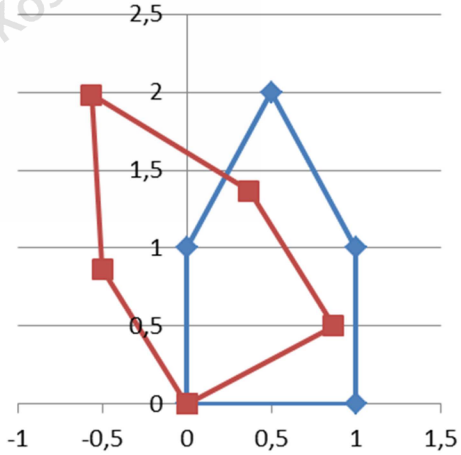
$$P' = P * \begin{bmatrix} 1 & 0 & 0 \\ h & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = P * \boldsymbol{H}$$



Rys. 67. Pochylenie z parametrem h

4. Obrót o kąt *a*:

$$P' = P * \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} = P * \boldsymbol{R}$$



Rys. 68. Obrót o kąt α

Przekształcenia można składać i przedstawić w postaci wypadkowej macierzy utworzonej przez wymnożenie macierzy przekształceń składowych. Np. dla dwóch przekształceń zapisanych w macierzach $\boldsymbol{M_1}$ *i* $\boldsymbol{M_2}$:

$$P' = P * \boldsymbol{M_1}$$

$$P'' = P' * \boldsymbol{M_2}$$

$$P'' = P' * \boldsymbol{M_2} = P * \boldsymbol{M_1 M_2} = P * \boldsymbol{M}, \qquad \boldsymbol{M} = \boldsymbol{M_1 M_2}$$

Składając cztery podstawowe przekształcenia afiniczne, można uzyskać dowolną macierz 3x2 (w wierszowym zapisie macierzy transformacji ostatnia kolumna macierzy jest niewykorzystywana). Złożenie tych przekształceń w odpowiedniej kolejności umożliwi również łatwe wyznaczenie parametrów tych przekształceń na podstawie dowolnej macierzy transformacji. Wymaga to rozwiązania układu 6 równań, którego niewiadomymi jest 6 parametrów transformacji podstawowych. Można zauważyć, że zakładając następującą kolejność przekształceń: skalowanie, pochylenie, obrót i przesunięcie, uzyska się macierz transformacji:

$$\begin{aligned} \boldsymbol{M} = \boldsymbol{SHRT} &= \begin{bmatrix} m_{00} & m_{01} & 0 \\ m_{10} & m_{11} & 0 \\ m_{20} & m_{21} & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ s_y h & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ t_x & t_y & 1 \end{bmatrix} \\ &= \begin{bmatrix} s_x \cos\alpha & s_x \sin\alpha & 0 \\ s_y(h\cos\alpha - \sin\alpha) & s_y(h\sin\alpha + \cos\alpha) & 0 \\ t_x & t_y & 1 \end{bmatrix} \end{aligned} \tag{7.1}$$

Parametry można wyznaczać od końca i cofać poznane przekształcenia. Przesunięcie wykonane jako ostatnie przekształcenie, nie zmienia macierzy SHR o rozmiarze 2x2, a jedynie ostatni wiersz macierzy. Z tego wiersza można wprost odczytać współrzędne wektora [tx, ty] przesunięcia transformowanego obiektu. Można cofnąć to przesunięcie, mnożąc przez macierz przesunięcia o wektor [-tx, -ty].

$$\boldsymbol{M'} = \boldsymbol{SHRTT^{-1}} = \boldsymbol{SHR} = \begin{bmatrix} s_x \cos\alpha & s_x \sin\alpha & 0 \\ s_y(h\cos\alpha - \sin\alpha) & s_y(h\sin\alpha + \cos\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Kąt obrotu α może być określony na podstawie:

$$tg\alpha = \frac{m'_{01}}{m'_{00}}$$

Teraz można cofnąć obrót, mnożąc przez macierz obrotu o kąt –α:

$$\boldsymbol{M''} = \boldsymbol{M'R^{-1}} = \boldsymbol{SH} = \begin{bmatrix} s_x & 0 & 0 \\ s_y h & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$s_x = m''_{00}$$

$$s_y = m''_{11}$$

$$h = \frac{m''_{10}}{m''_{11}}$$

Warto zauważyć, że faktoryzacja SHRT macierzy możliwa jest również dla transformacji w przestrzeni 3D a dla wyższych wymiarów związana jest blisko z faktoryzacją SVD.

W zapisie kolumnowym macierze kolejnych przekształceń mnożą macierz transformacji z lewej strony. Kolejność przekształceń należy wówczas odczytywać od prawej do lewej. Ponieważ w zapisie kolumnowym macierz transformacji jest mnożona przez przekształcany wektor współrzędnych punktu, macierze podstawowych transformacji mają postać macierzy transponowanych w stosunku do zapisu wierszowego.

```
// Obiekt może podlegać transformacjom, które są zapamiętywane w macierzy Transform.
// Każdą macierz można przedstawić jako złożenie czterech podstawowych transformacji
// wykonanych w odpowiedniej kolejności: skalowania, pochylenia, obrotu i przesunięcia
// Parametry tych podstawowych transformacji zapamiętywane są w wektorach:
// Scale, Shear, Rotation i Origin (SHRT).
public Matrix Transform
{
    get
    {
        Matrix FTransform = new Matrix();
        FTransform.Translate(Origin.X, Origin.Y);
        FTransform.Rotate(Rotation);
        FTransform.Shear(Shear, 0);
        FTransform.Scale(Scale.X, Scale.Y);
        return FTransform;
    }
    // Na podstawie dowolnej macierzy przekształceń można wyznaczyć parametry SHRT
    // Wyznaczane są one od końca a macierz mnożona przez transformację odwrotną
    set
    {
        Origin.X = value.Elements[4];
        Origin.Y = value.Elements[5];
        value.Translate(-Origin.X, -Origin.Y, MatrixOrder.Append);
        double alpha = Math.Atan2(value.Elements[1], value.Elements[0]);
        Rotation = (float)(alpha * 180 / Math.PI);
        value.Rotate(-Rotation, MatrixOrder.Append);
        Scale.X = value.Elements[0];
        Scale.Y = value.Elements[3];
        Shear = value.Elements[2] / Scale.Y;
    }
}
```

Podczas grupowania tworzony jest nowy kształt - grupa, która staje się nowym rodzicem kształtów. Grupa może następnie podlegać dalszym transformacjom, które wpływają na jej podkształty. Z tego powodu, przy rozgrupowaniu, przed zmianą rodzica podkształtów na rodzica grupy i jej usunięciem, należy złożyć ich transformacje z transformacją grupy. Złożenie dwóch transformacji powoduje, że parametry SHRT kształtu przestają być prawidłowe, dlatego wykorzystywany jest setter właściwości *Transform* kształtów.

Dla obiektów klasy *TShape* możliwe są operacje transformacji punktu (np. wierzchołka) przez macierz transformacji kształtu, jak również operacja odwrotna – obliczenie współrzędnych punktu przed transformacją na podstawie współrzędnych punktu po transformacji. Umożliwiają one m.in. manipulację interaktywną transformacji.

```
    public PointF TransformPoint(PointF p)
    {
        PointF[] pts = new PointF[] { p };
        Transform.TransformPoints(pts);
        return pts[0];
    }
    public PointF UnTransformPoint(PointF p)
    {
        PointF[] pts = new PointF[] { p };
        Matrix invTransform = Transform.Clone();
        invTransform.Invert();
        invTransform.TransformPoints(pts);
        return pts[0];
    }
```

Na podstawie współrzędnych wierzchołków kształtu i jego podkształtów wyznaczany jest prostokąt obejmujący. Jest to najmniejszy prostokąt bez transformacji, który zawiera wszystkie wierzchołki kształtu i prostokąty obejmujące podkształtów. Ułatwia on ocenę widoczności kształtu, jego kolizji z innymi kształtami, promieniami światła itp.

```
// Każdy obiekt zawiera się w prostokącie obejmującym BBox, który uwzględnia
// wymiary własne obiektu i przetransformowane BBox'y wszystkich dzieci obiektu.
// W literaturze często można spotkać określenie AABB (Axis Aligned Bounding Box).
// Niekiedy również używana jest kula obejmująca (Bounding Sphere)
RectangleF bbox;
public RectangleF BBox
{
    get
    {
        for (int i = 0; i < Vertices.Count; i++)
        {
            var rc = new RectangleF(Vertices[i], Size.Empty);
            bbox = (i == 0) ? rc : RectangleF.Union(bbox, rc);
        }
        bool needInit = bbox.IsEmpty;
        for (int i = 0; i < Children.Count; i++)
        {
            TShape child = Children[i];
            var childBBox = child.BBox;
            PointF lbn = childBBox.Location;
            PointF rtf = childBBox.Location + childBBox.Size;
            for (int j = 0; j < 4; j++)
            {
                PointF v = lbn;
                if ((j & 1) != 0) v.X = rtf.X;
                if ((j & 2) != 0) v.Y = rtf.Y;
                v = child.TransformPoint(v);
                if (needInit)
                {
                    bbox = new RectangleF(v, Size.Empty);
                    needInit = false;
                }
                else
                    bbox = RectangleF.Union(bbox, new RectangleF(v, Size.Empty));
            }
        }
        return bbox;
    }
}
```

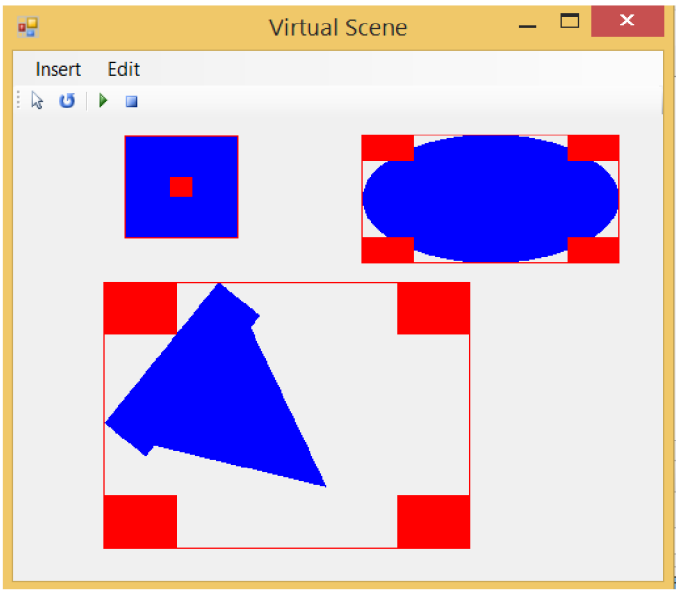
Prostokąt obejmujący został wykorzystany również do identyfikacji kształtu na podstawie punktu kliknięcia myszką. Procedura wykorzystuje metodę *ShapeAt* kształtu, wyznaczającą podkształt, którego prostokąt obejmujący zawiera podany punkt w układzie współrzędnych rodzica kształtu - współrzędnych świata w przypadku *Root*. W trybie manipulacji kształtami, zaznaczalne są wyłącznie kształty będące bezpośrednio podrzędnymi w stosunku do korzenia hierarchii *Root*. Dostęp do podkształtów grupy możliwy jest dopiero po rozgrupowaniu.

```
public TShape ShapeAt(PointF pos)
{
    pos = UnTransformPoint(pos);
    for (int i = Children.Count - 1; i >= 0; i--)
    {
        TShape child = Children[i];
        if (child.BBox.Contains(child.UnTransformPoint(pos))) return child;
    }
    return null;
}
```

Sama klasa kształtów jest abstrakcyjna. Tworzone instancje są typów, które po niej dziedziczą. Specjalizowanymi klasami kształtów są *TRectangle*, *TEllipse*, *TTriangle*, które nadpisują konstruktor w celu utworzenia listy wierzchołków i procedurę rysowania danego typu kształtu, np. dla trójkąta:

```
public TTriangle()
{
    Vertices.Add(new PointF(-1, -1));
    Vertices.Add(new PointF(1, -1));
    Vertices.Add(new PointF(0, 1));
}
protected override void DrawShape(Graphics context)
{
    context.FillPolygon(Brush, Vertices.ToArray());
}
```

Do interaktywnej zmiany transformacji wykorzystana została klasa gumki *TRubber*, która również dziedziczy po kształcie. W momencie zaznaczenia kształtu do edycji, tworzona jest instancja gumki, reprezentowana w postaci prostokąta obejmującego i markerów, będących podkształtami gumki. Markery umożliwiają zmianę parametrów transformacji. Sama gumka staje się kształtem podrzędnym w stosunku do modyfikowanego kształtu tak, by jego transformacje wpływały również na gumkę. Położenie i liczba markerów zależy od trybu gumki, określonego przez rodzaj edytowanych transformacji (Rys. 69).



Rys. 69. Zmiana transformacji obiektów przy użyciu gumek

Przy odznaczeniu kształtu, gumka musi być usunięta, dlatego edytor posiada listę istniejących gumek, która pośrednio informuje o zaznaczonych kształtach. W momencie kliknięcia na gumkę (czyli również na zaznaczony kształt), wywoływana jest metoda *MouseDown* gumki, która określa marker, który został wybrany do modyfikacji, oraz zapamiętuje współrzędne punktu kliknięcia myszy *StartPos*:

```
Point StartPos;
TShape Selected;
public void MouseDown(Point pos)
{
    StartPos = pos;
    var posF = Control.UnTransformPoint(Root.UnTransformPoint(pos));
    Selected = ShapeAt(posF);
}
```

Reakcja na przeciągnięcie myszki zależy od trybu edycji transformacji. W przypadku skalowania z przesunięciem, markery rozstawione są w rogach prostokąta obejmującego kształt. Jeśli nie został zaznaczony żaden z markerów, kształt ulega przesunięciu - współrzędne zaczepienia obiektu *Origin* są zmieniane o różnicę transformacji odwrotnych położeń myszki w momencie kliknięcia i aktualnego jej położenia. W przypadku kliknięcia markera, zakłada się, że modyfikowany marker (wierzchołek *lbn*) zawsze wskazuje na ten sam róg prostokąta obejmującego kształt podlegający nowej transformacji, natomiast naprzeciwległy marker (wierzchołek *rtf*) nie zmienia swojego położenia. Podczas ruchu myszki można zatem określić nowe położenie P wierzchołka *lbn* gumki w układzie współrzędnych obiektu. Można zapisać równania:

$$\begin{cases} P * SHRT = lbn * S'HRT' \\ rtf * SHRT = rtf * S'HRT' \end{cases}$$

$$\begin{cases} P * SHR + T = lbn * S'HR + T' \\ rtf * SHR + T = rtf * S'HR + T' \end{cases}$$

gdzie S' i T' to szukane parametry nowej skali i przesunięcia dla obiektu. Odejmując równania otrzymujemy:

$$(rtf - P) * SHR = (rtf - lbn) * S'HR$$

$$(rtf - P) * S = (rtf - lbn) * S'$$

$$S' = S * \frac{rtf - P}{rtf - lbn} \tag{7.2}$$

Parametr T' można wyznaczyć np. z drugiego równania:

$$rtf * SHR + T = rtf * S'HR + T'$$

$$T' = T + rtf * SHRT - rtf * S'HRT \tag{7.3}$$

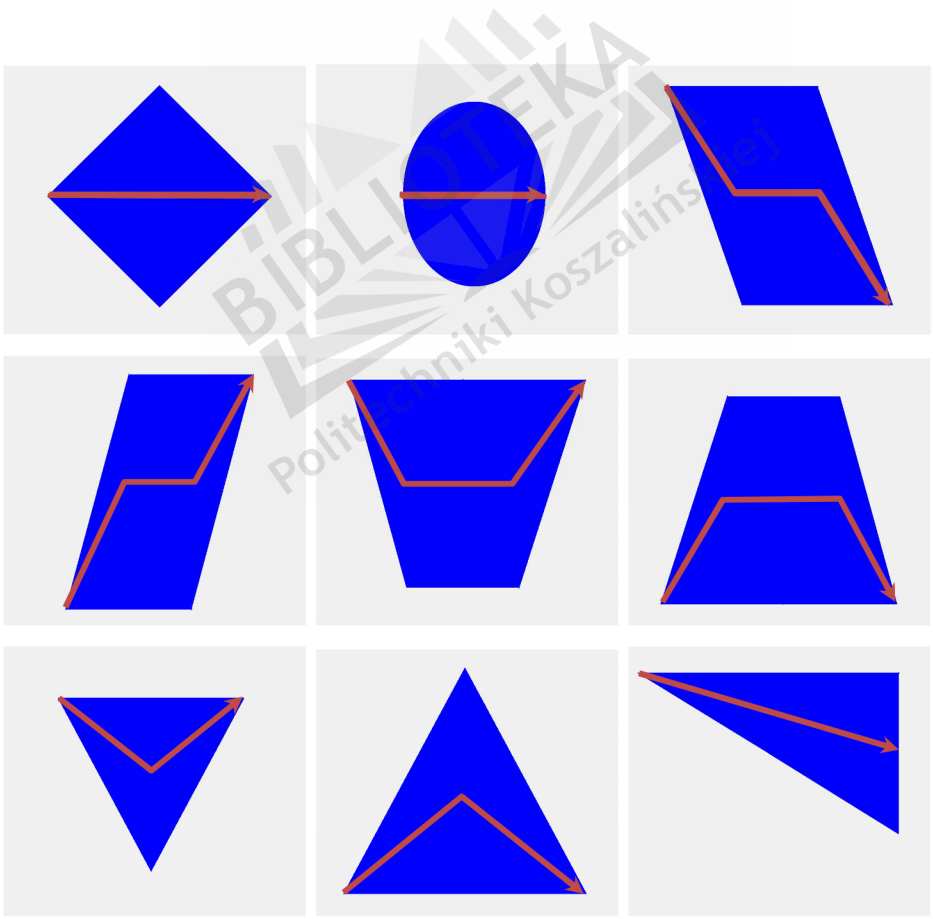
Przy czym równania (7.2) i (7.3) są spełnione zarówno dla współrzędnych x, jak i y.

W trybie obrotu, dla gumki tworzony jest tylko jeden marker w punkcie zaczepienia kształtu. Kliknięcie i przesunięcie myszki powoduje zmianę parametru obrotu kształtu o sumę różnic położeń przy kliknięciu i aktualnego położenia myszy po osi X i Y. Przy przeciąganiu myszki wywoływana jest metoda *MouseMove* gumki:

```
public void MouseMove(Point pos)
{
    if (Mode == TRubberMode.Scaling)
    {
        if (Selected == null)
        {
            PointF posF = Root.UnTransformPoint(pos);
            PointF startPosF = Root.UnTransformPoint(StartPos);
            Control.Origin.X += posF.X - startPosF.X;
            Control.Origin.Y += posF.Y - startPosF.Y;
            StartPos = pos;
        }
        else
        {
            PointF lbn = Selected.Origin;
            PointF rtf = Children[Children.IndexOf(Selected) ^ 3].Origin;
            PointF lbnNew = Root.UnTransformPoint(pos);
            lbnNew = Control.UnTransformPoint(lbnNew);
            PointF rtfPos = Control.TransformPoint(rtf);
            Parent.Scale.X *= (rtf.X - lbnNew.X) / (rtf.X - lbn.X);
            Parent.Scale.Y *= (rtf.Y - lbnNew.Y) / (rtf.Y - lbn.Y);
            PointF rtfPosNew = Control.TransformPoint(rtf);
            Parent.Origin.X += rtfPos.X - rtfPosNew.X;
            Parent.Origin.Y += rtfPos.Y - rtfPosNew.Y;
        }
    }
    else if (Mode == TRubberMode.Rotating)
    {
        float angle = 0;
        angle += pos.X - StartPos.X;
        angle += pos.Y - StartPos.Y;
```

```
                Parent.Rotation += angle;
                StartPos = pos;
            }
            Invalidate();
        }
```

Moduł umożliwia naukę rozpoznawania kształtów zakodowanych jako kolumny o zmieniającej się pozycji horyzontalnej i wertykalnej. Informacja ta przekazywana jest poprzez lokalizację wirtualnego źródła dźwięku. O ile stosunkowo dobrze rozpoznawalne jest położenie źródła dźwięku w poziomie, rozpoznawanie położenia w pionie jest znacznie trudniejsze i może wymagać treningu. Położenie środków kolumn wyznacza pewną charakterystyczną dla kształtu krzywą, będącą efektem jego szkieletyzacji. Rys. 70 przedstawia przykłady prostych kształtów z zaznaczoną krzywą zmiany położenia wirtualnego źródła dźwięku. Warto podkreślić, że niektóre kształty, np. równoległoboki i trapezy mogą różnić się wyłącznie pozycją wertykalną źródła.



Rys. 70. Przykłady kształtów z zaznaczonymi (na czerwono) ścieżkami przemieszczenia wirtualnego źródła dźwięku

Na Katedrze Systemów Multimedialnych i Sztucznej Inteligencji przeprowadzono eksperyment, w którym wzięło udział 5 osób. Osoby te poddano nauce rozpoznawania

kształtów jak na Rys. 70 i ich położenia względem ekranu. Naukę przeprowadzono dla następującej konfiguracji modułu:

- Instrument muzyczny: saksofon
- Zakres tonalny: 36 – 84 (standard MIDI)
- Proporcje ekranu 1:1
- Kąt elewacji i azymutu systemu dźwiękowego: od -45 do 45
- Liczba kolumn reprezentujących obiekt: 16
- Wysokość figur względem ekranu: około 70%
- Okres odtwarzania dźwięku: 1s

Czas nauki każdej osoby wynosił 2 godz. Uczącym się, oprócz informacji o mechanizmie generacji dźwięku, przekazano następujące zalecenia:

- Rozpocząć naukę od rozpoznawania położenia źródła dźwięku punktowego
- Naukę rozpoznawania kształtu figur rozpocząć dopiero po opanowaniu umiejętności lokalizacji źródła dźwięku
- W czasie nauki starać się najpierw kojarzyć melodię z kształtem, a następnie próbować dokonywać analizy kształtu ścieżki przemieszczania się źródła dźwięku i jej położenia względem obrazu (czerwone linie na figurach z Rys. 70)
- Skupić się na powtarzaniu tych kształtów, które sprawiają największe trudności
- W czasie nauki starać się zapamiętać z jednej strony melodię poszczególnych kształtów, a z drugiej - rozróżniać figury o podobnych kształtach (koło i elipsa) lub podobnej melodii (trójkąt skierowany wierzchołkiem ku dołowi i ku górze)

W czasie nauki zmieniano docelowo zarówno rodzaje figur, jak i ich położenie. Bezpośrednio po dwóch godzinach nauki przeprowadzono test polegający na próbie rozpoznania kształtów obiektów i ich położenia. Pozycję w kierunku pionowym wyznaczano korzystając z określeń: góra, środek i dół, a pozycję w kierunku poziomym: lewo, środek, prawo. W ten sposób rozpoznawano 9 figur, z których każda mogła przyjąć jedną z 9 pozycji. Nie limitowano czasu rozpoznania, ale średnio zawierał się on w przedziale 6-7 s. Średnia skuteczność rozpoznania kształtu figur dla testu zawierającego 50 prób, wyniosła około 75%. Trudno ocenić, czy otrzymany wynik może być satysfakcjonujący i czy może ulec istotnej poprawie w przypadku długotrwałej nauki oraz np. indywidualizacji funkcji HRTF. Pokazuje on jednak możliwości percepcji kształtu opartej o analizę cech dźwięku przestrzennego.

W czasie eksperymentu oraz jego przygotowań zauważono również inne zjawiska, które mogą rzutować na skuteczność nauki rozpoznawania kształtów proponowaną metodą. Najniższą rozróżnialność kształtów osiągnięto dla koła i elipsy – figury te były dobrze rozpoznawane względem innych figur, ale nieskuteczne było rozróżnianie tych figur pomiędzy sobą. Skuteczność rozpoznawania kształtów była tym gorsza, im

mniejsza była wysokość obiektu. Jest to prawdopodobnie spowodowane malejącym zakresem tonalnym dźwięku reprezentującego obiekt oraz ogólnie niską rozróżnialnością położenia źródła dźwięku w pionie. Na podstawie przeprowadzonych eksperymentów można stwierdzić, że dokładność lokalizacji położenia źródła dźwięku jest większa dla kierunku poziomego niż pionowego, ale również zauważalna jest różnica wrażliwości na zmianę położenia źródła w obszarach powyżej i poniżej linii horyzontu. Należy zwrócić uwagę, że detekcja położenia źródła dźwięku ma bardzo istotny wpływ na możliwość analizy kształtu. Z tego powodu jednym z najbardziej istotnych kierunków rozwoju proponowanej metody powinno być doskonalenie selektywności rozdzielczości przestrzennej dźwięku, m.in. poprzez indywidualny pomiar charakterystyki HRTF.

Zauważono również, że nabyte umiejętności ulegają dość szybkiej degradacji – dzień po pierwszym eksperymencie skuteczność rozpoznawania była znacznie niższa. Jednak odtworzenie stanu umiejętności trwało znacznie krócej niż proces uczenia.

7.1 Podsumowanie

Opisany moduł może być wykorzystany do nauki rozponania kształtu figur i ich położenia względem wirtualnej sceny. Dzięki wykorzystaniu transformacji afinicznych i hierarchicznej struktury danych możliwe jest tworzenie złożonych scen 2D, które są poddawane konwersji przez moduł generowania dźwięku przestrzennego. Transformacje kształtów mogą być łatwo modyfikowane za pomocą interaktywnych „gumek" wykorzystujących myszkę lub touchpad. Podwójne kliknięcie na kształt wywołuje okno jego właściwości, umożliwiając wprowadzenie i modyfikację parametrów transformacji geometrycznej. Sceny mogą być zapisywane i odczytywane z plików, domyślnie umieszczonych w podkatalogu *Scenes* Systemu.

8. Wnioski końcowe

W pracy przedstawiono metodę wspomagania orientacji przestrzennej osób niewidomych opartą na transformacji obrazów 2D do postaci dźwięku przestrzennego oraz wyniki wstępnych badań weryfikujących poprawność przyjętej tezy. Metoda zawiera etap akwizycji obrazów, przetwarzania wstępnego, segmentacji oraz transformacji kształtu analizowanego obiektu do postaci dźwięku a także wspomaganie rozpoznania obiektu z użyciem sztucznej sieci neuronowej. W pracy wykazano, że zaproponowana metoda transformacji obraz – dźwięk może być skuteczna w rozpoznawaniu kształtu obiektów, ich położenia oraz szacowania odległości do obiektu i jego wielkości, a więc może wspomagać orientację przestrzenną osób niewidomych.

Zaprezentowany sposób przetwarzania obrazu na dźwięk przestrzenny umożliwia przekazywanie informacji o kształcie i położeniu centralnego obiektu sceny. Akwizycja obrazów, w odróżnieniu od systemów stereowizyjnych odbywa się za pomocą jednej kamery. Może to mieć wpływ na obniżenie kosztów i uproszczenie kalibracji systemu. Rozpoznanie kształtu i położenia obiektu realizowane jest na podstawie analizy dźwięku skorelowanego z rozmiarami kolumn, na które dzielony jest obiekt, i ich położeniem. Taki sposób reprezentacji kształtu i położenia jest jednoznaczny i jest nowatorski. Inne systemy wykorzystujące segmentację umożliwiają przede wszystkim detekcję wielkości i położenia obiektów. Wykorzystanie do syntezy dźwięków próbek instrumentów muzycznych, w standardzie zgodnym z MIDI, ułatwia uzyskanie harmonii dźwięków, co jest bardzo istotne w przypadku częstego i długotrwałego narażenia użytkownika na jego oddziaływanie.

Opracowanie nowych algorytmów segmentacji pozwoliło na poprawę jej jakości przy stosunkowo niskim koszcie obliczeniowym.

Zastosowanie algorytmu analizy czynników głównych PCA w module rozpoznawania obiektów sceny umożliwiło zmniejszenie liczby wejść SSN. Pozwoliło to również skrócić czas ich uczenia. Moduł rozpoznawania obiektów pełni funkcję wspomagającą w stosunku do rozpoznawania opartego o analizę dźwięku.

Wynikiem pracy jest również program komputerowy, w którym zaimplementowano opisaną metodę oraz algorytmy przetwarzania. Optymalizacja algorytmów i kodu programu pozwoliła skrócić czas całkowitego przetwarzania (od akwizycji obrazu do generacji dźwięku) i obecnie jest on znacznie krótszy od czasu niezbędnego do analizy dźwięku przez użytkownika, co oznacza, że przetwarzania są realizowane w czasie rzeczywistym.

Z przeprowadzonych badań wynika, że rozpoznawanie kształtów i ich położenia, nawet w przypadku bardzo dobrej segmentacji, wymaga treningu. Metoda konwersji wymaga oswojenia się i zapamiętania kluczowych wzorców, podobnie jak np. w technice alfabetu Braille'a. Dlatego stworzono moduł do nauki rozpoznawania kształtu wirtualnych obiektów oraz ich położenia. Zawiera on edytor grafiki 2D umożliwiający tworzenie, edytowanie oraz konwertowanie dowolnych kształtów geometrycznych na dźwięk przestrzenny.

Szczególnie trudnym zadaniem okazało się rozpoznawanie położenia źródła dźwięku w pionie. Dla małych obiektów różnica położeń współrzędnych pionowych środków kolumn jest słabo rozróżnialna. Dlatego naturalnym kierunkiem rozwoju tej metody powinna być indywidualizacja funkcji HRTF, poprawiająca rozdzielczość rozpoznawania położenia źródeł dźwięku.

Założone cele pracy zostały zrealizowane, a teza o możliwości analizy kształtu, położenia i szacowania odległości osoby niewidomej do obiektu wykorzystującej metodę transformacji obrazów 2D do postaci dźwięku przestrzennego została potwierdzona eksperymentalnie.

A. Instrumenty dostępne w standardzie General MIDI

Piano (0-7)	Acoustic Grand Piano	Bright Acoustic Piano	Electric Grand Piano	Honky-tonk Piano	Electric Piano 1	Electric Piano 2	Harpsi-chord	Clavinet
Chromatic Percussion (8-15)	Celesta	Glocken-spiel	Music Box	Vibraphone	Marimba	Xylophone	Tubular Bells	Dulcimer
Organ (16-23)	Drawbar Organ	Percussive Organ	Rock Organ	Church Organ	Reed Organ	Accordion	Harmonica	Tango Accordion
Guitar (24-31)	Acoustic Guitar (nylon)	Acoustic Guitar (steel)	Electric Guitar (jazz)	Electric Guitar (clean)	Electric Guitar (muted)	Overdriven Guitar	Distortion Guitar	Guitar Harmonics
Bass (32-39)	Acoustic Bass	Electric Bass (finger)	Electric Bass (pick)	Fretless Bass	Slap Bass 1	Slap Bass 2	Synth Bass 1	Synth Bass 2
Strings (40-47)	Violin	Viola	Cello	Contrabass	Tremolo Strings	Pizzicato Strings	Orchestral Harp	Timpani
Ensemble (48-55)	String Ensemble 1	String Ensemble 2	Synth Strings 1	Synth Strings 2	Choir Aahs	Voice Oohs	Synth Choir	Orchestra Hit
Brass (56-63)	Trumpet	Trombone	Tuba	Muted Trumpet	French Horn	Brass Section	Synth Brass 1	Synth Brass 2
Reed (64-71)	Soprano Sax	Alto Sax	Tenor Sax	Baritone Sax	Oboe	English Horn	Bassoon	Clarinet
Pipe (72-79)	Piccolo	Flute	Recorder	Pan Flute	Blown bottle	Shakuhachi	Whistle	Ocarina
Synth Lead (80-87)	square	sawtooth	calliope	chiff	charang	voice	fifths	bass & lead
Synth Pad (88-95)	new age	warm	polysynth	choir	bowed	metallic	halo	sweep
Synth Effects (96-103)	rain	soundtrack	crystal	atmosphere	brightness	goblins	echoes	sci-fi
Ethnic (104-111)	Sitar	Banjo	Shamisen	Koto	Kalimba	Bagpipe	Fiddle	Shanai
Percussive (112-119)	Tinkle Bell	Agogo	Steel Drums	Woodblock	Taiko Drum	Melodic Tom	Synth Drum	Reverse Cymbal
Sound Effects (120-127)	Guitar Fret Noise	Breath Noise	Seashore	Bird Tweet	Telephone Ring	Helicopter	Applause	Gunshot

Bibliografia

[1] S. Meers i K. Ward, „A vision system for providing 3D perception of the environment via transcutaneous electro-neural stimulation," w *Proc. of 8th International conference on information visualization*, 2004.

[2] H. Sumiya, „Distance Feedback Travel Aid Haptic Display Design," w *Mobile Robots: Perception & Navigation*, S. Kolski, Red., Pro Literatur Verlag, Germany, 2007, pp. 395-412.

[3] J. M. Benjamin, N. A. Ali i A. F. Schepis, „A Laser Cane for the Blind," *Proceedings of the San Diego Biomedical Symposium,* tom 12, pp. 53-57, 1973.

[4] S. Shoval, J. Borenstein i Y. Koren, „Mobile Robot Obstacle Avoidance in a Computerized Travel Aid for the Blind," *Proceedings of the 1994 IEEE International Conference on Robotics and Automation,* pp. 2023-2029, 8-13 May 1994.

[5] J. Borenstein i I. Ulrich, „The GuideCane — A Computerized Travel Aid for the Active Guidance of Blind Pedestrians," *Proceedings of the IEEE International Conference on Robotics and Automation,* pp. 1283-1288, 21-27 Apr 1997.

[6] S. Shoval, I. Ulrich i J. Borenstein, „NavBelt and the Guide-Cane [obstacle-avoidance systems for the blind and visually impaired]," *Robotics & Automation Magazine, IEEE,* tom 10, nr 1, pp. 9-20, 2003.

[7] N. Pressey, „Mowat Sensor," *Focus,* tom 11, nr 3, pp. 35-39, 1977.

[8] D. Bissitt i A. D. Heyes, „An Application of Biofeedback in the Rehabilitation of the Blind," *Applied Ergonomics,* tom 11, nr 1, pp. 31-33, 1980.

[9] L. Kay, „Auditory Perception of Objects by blind persons using bio-acoustic high resolution air sonar," *JASA,* tom 107, nr 6, pp. 3266-3275, June 2000.

[10] L. Kay, „A Sonar Aid to Enhance Spatial Perception of the Blind: Engineering Design and evaluation," *Radio and Electronic Engineer,* tom 44, nr 11, pp. 605-627, 1974.

[11] P. Meijer, An experimental system for auditory image representations, IEEE Transactions on Biomedical Engineering 39, pp. 112-121, 1992.

[12] J. L. Gonzalez-Mora, A. Rodriguez-Hernandez, L. F. Rodriguez-Ramos, L. Diaz-Saco i N. Sosa, „Development of a new space perception system for blind people, based on the creation of a virtual acoustic space," *Proceedings of the International Work Conference on Artificial and Natural Networks (IWANN),* p. 321–330, 1999.

[13] M. Bujacz i P. Strumiłło, „Dźwiękowa prezentacja otoczenia niewidomym: prototyp urządzenia Naviton," *Przegląd Telekomunikacyjny,* tom 8, pp. 733-734, 2011.

[14] G. Balakrishnan, G. Sainarayanan, R. Nagarajan i S. Yaacob, „A Stereo Image Processing System for Visually Impaired," *International Journal of Information and Communication Engineering,* tom 2, nr 3, p. 136, 2008.

[15] B. Deville, G. Bologna, M. Vinckenbosch i T. Pun, „See ColOr: Seeing Colours with an Orchestra," *Human Machine Interaction,* tom LNCS 5440, p. 251–279, 2009.

[16] L. Busin, N. Vandenbroucke i L. Macaire, „Advances in Imaging and Electron Physics," tom 151, pp. 65-168, 2008.

[17] R. S. Hunter, „Photoelectric Color Difference Meter," *Journal of the Optical Society of America,* tom Vol. 48, nr 12, pp. 985-993, 1958.

[18] R. Świta, „Porównywanie pikseli obrazów kolorowych na potrzeby segmentacji," w *ICT Young 2012*, Gdańsk, 2012.

[19] O. Warusfel, „LISTEN HRTF database," http://recherche.ircam.fr/equipes/salles/listen, 2003.

[20] W. H. Press, S. A. Teukolsky, V. W. T. i B. P. Flannery, Numerical Recipes in FORTRAN 77, Cambridge: Cambridge University Press, 1992.

[21] E. Catmull i R. Rom, „A class of local interpolating splines," *Computer Aided Geometric Design ,* tom 1, pp. 317-326, 1974.

[22] C. de Boor, A Practical Guide to Splines, New York: Springer-Verlag, 1978.

[23] P. J. Hartley i C. J. Judd, „Parametrization of Bezier Type B-spline curves and surfaces," *Computer Aided Design,* tom 10, nr 2, pp. 130-134, 1978.

[24] P. J. Hartley i C. J. Judd, „Parametrization and shape of B-spline curves for CAD," *Computer Aided Design,* tom 12, nr 5, pp. 235-238, 1980.

[25] I. J. Schoenberg i A. Whitney, „On Pólya frequency functions III," *Transactions of the American Mathematical Society,* tom 74, pp. 246-259, 1953.

[26] M. Unser, „Splines: a perfect fit for signal and image processing," *IEEE Signal Processing Magazine,* tom 16, pp. 22-38, 1999.

[27] G. Farin, NURB Curves and Surfaces, A. K. Peters, 1999.

[28] B. Phong, „Illumination for computer generated pictures," *Communications of the ACM,* tom 18, nr 6, pp. 311-317, 1975.

[29] G. Wyszecki i W. S. Stiles, Color science, New York: Wiley, 1982.

[30] Q. Zaidi, B. Spehar i D. J., „Color constancy invariegated scenes: role of low-level mechanisms in discounting illumination changes," *J. Opt. Soc. Am.,* tom 14, p. 2608–2621, 1997.

[31] G. Buchsbaum, „A spatial processor model for object colour perception," *Journal of The Franklin Institute,* tom 310, p. 1–26, 1980.

[32] N. Lloyd i D. B. Trefethen III, Numerical Linear Algebra, SIAM: Society for Industrial and Applied Mathematics, 1997.

[33] N. J. Higham, „QR Factorization," w *Accuracy and Stability of Numerical Algorithms*, Manchester, SIAM, 2002, pp. 353-378.

[34] G. H. Golub i C. F. Van Loan, „The QR Factorization," w *Matrix Computations 4th edition*, Baltimore, The Johns Hopkins University Press, 2013, pp. 246-260.

[35] G. H. Golub i C. F. Van Loan, „The Practical QR Algorithm," w *Matrix Computations 4th edition*, Baltimore, The Johns Hopkins University Press, 2013, pp. 385-393.

[36] J. H. Wilkinson, The algebraic eigenvalue problem, Oxford: Oxford University Press, 1965.

[37] I. Jolliffe, Principal Component Analysis 2nd ed., New York: Springer, 2002.

[38] R. Świta i Z. Suszyński, „Processing of thermographic sequence using Principal Component Analysis," *Measurement Automation Monitoring,* tom 61, nr 06, pp. 215-218, 2015.

[39] J. Korbicz, J. Kościelny, Z. Kowalczuk i W. Cholewa, Fault Diagnosis - Models, Artificial Intelligence, Applications, Berlin: Springer-Verlag, 2004.

[40] A. K. Cline i I. S. Dhillon, „Computation of the Singular Value Decomposition," w *Handbook of Linear Algebra*, CRC Press, 2006, pp. 45.1-45.13.

[41] G. Golub i W. Kahan, „Calculating the Singular Values and Pseudo-inverse of a Matrix," *SIAM Journal on Numerical Analysis,* tom 2, nr 2, pp. 205-224, 1965.

[42] J. Demmel i W. Kahan, „Accurate Singular Values of Bidiagonal Matrices," *SIAM J. Sci. Stat. Comput.,* tom 11, nr 5, pp. 873-912, 1990.

[43] J. W. Demmel, Applied Numerical Linear Algebra, Philadephia: SIAM, 1997.

[44] L. N. Trefethen i D. Bau III, „Computing the SVD," w *Numerical Linear Algebra*, SIAM, 1997, pp. 234-240.

[45] J. C. G. Jacob, „Uber ein leichtes Verfahren die in der Theorie der Sacularstorungen vorkommenden Gleichungen numerisch aufzulosen," *Crelle's Journal für reine und angew. Math.,* tom 30, p. 51–95, 1846.

[46] J. Demmel i K. Veselić, „Jacobi's Method is More Accurate than QR," *SIAM Journal on Matrix Analysis and Applications,* tom 13, nr 4, p. 1204–1245, 1992.

[47] Z. Drmač i K. Veselić, „New fast and accurate Jacobi SVD algorithm: I i II," *SIAM. J. Matrix Anal. & Appl.,* tom 29, nr 4, p. 1322–1362, 2008.

[48] G. Sewell, Computational Methods of Linear Algebra, New Jersey: John Wiley & sons, 2005.

[49] E. Forgy, „Cluster Analysis of Multivariate Data: Efficiency versus Interpretability of Classification," *Biometrics,* tom 21, nr 3, pp. 768-769, 1965.

[50] J. MacQueen, „Some methods for classification and analysis of multivariate observations," *Proc. Fifth Berkeley Symp. on Math. Statist. and Prob.,* tom 1, pp. 281-297, 1967.

[51] S. P. Lloyd, „Least squares quantization in PCM," *IEEE Transactions on Information Theory ,* tom 28 , nr 2, pp. 129-137, 1982.

[52] Wu, X.; Kumar, V.; Quinlan, J.R.; Ghosh, J.; Yang, Q.; Motoda, H.; McLachlan, G. J.; Ng, A. F. M.; Liu, B., „Top 10 algorithms in data mining," *Knowledge and Information Systems,* tom 14, nr 1, pp. 1-37, 2008.

[53] R. Świta i Z. Suszyński, „New method of segmentation of thermal image sequences," w *Microelectronic Materials and Technologies tom 2*, Koszalin, Wydawnictwo Politechniki Koszalińskiej, 2012, pp. 31-44.

[54] R. Świta i Z. Suszyński, „Segmentacja sekwencji obrazów metodą korelacyjną," *Pomiary Automatyka Kontrola,* tom 7, pp. 680-684, 2013.

[55] J. C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms, New York: Plenum Press, 1981.

[56] J. M. Peña, J. A. Lozano i P. Larrañaga, „An empirical comparison of four initialization methods for the K-Means algorithm," *Pattern Recognition Letters,* tom 20, nr 10, pp. 1027-1040 , 1999.

[57] I. Katsavounidis, C.-C. Jay Kuo i Z. Z., „A new initialization technique for generalized Lloyd iteration," *Signal Processing Letters, IEEE,* tom 1, nr 10, pp. 144 - 146, 1994.

[58] D. Arthur i V. S., „k-means++: the advantages of careful seeding," *SODA '07 Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms,* pp. 1027-1035, 2007.

[59] S. J. Redmond i C. Heneghan, „A method for initialising the K-means clustering algorithm using kd-trees," *Pattern Recognition Letters,* tom 28, nr 8, p. 965–973, 2007.

[60] R. Świta i Z. Suszyński, „Cluster segmentation of thermal image sequences using kd-tree structure," *International Journal of Thermophysics,* tom 35, nr 12, pp. 2374-2387, 2014.

[61] R. Świta i Z. Suszyński, „Inicjalizacja segmentacji k-means uwzględniająca rozkład gęstości pikseli," *Zeszyty Naukowe Wydziału Elektroniki i Informatyki Politechniki Koszalińskiej,* tom 6, pp. 89-98, 2014.

[62] C. Lantuéjoul i H. Digabel, „Iterative algorithms," *Proc. 2nd European Symp. Quantitative Analysis of Microstructures in Material Science, Biology and Medicine,* p. 85–89, 1978.

[63] F. Meyer i S. Beucher, „The morphological approach to segmentation: the watershed transformation," *Mathematical Morphology in Image Processing,* p. 433–481, 1993.

[64] J. Cousty, G. Bertrand, L. Najman i M. Couprie, „Watershed Cuts: Minimum Spanning Forests and the Drop of Water Principle," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* tom 31(8), p. 1362–1374, 2009.

[65] R. Świta i Z. Suszyński, „Hybrid watershed and region grow segmentation of color images," *Zeszyty Naukowe Wydziału Elektroniki i Informatyki Politechniki*

Koszalińskiej, tom 3, pp. 111-122, 2011.

[66] M. W. Rand, „Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association,* tom 66, nr 336, p. 846–850, 1971.

[67] M. Meila, „Comparing Clusterings by the Variation of Information," *Learning Theory and Kernel Machines,* p. 173–187, 2003.

[68] J. Korbicz i A. Obuchowicz, Sztuczne sieci neuronowe: podstawy i zastosowania, Akademicka Oficyna Wydawnicza PLJ, 1994.

[69] S. K. Roffler i R. A. Butler, „Factors That Influence the Localization of Sound in the Vertical Plane," *J. Acoust. Soc. Am.,* tom 43, p. 1255, 1968.

[70] V. R. Algazi, R. O. Duda, D. M. Thompson i C. Avendano, „The CIPIC HRTF database," *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics,* p. 99–102, 2001.

[71] W. G. Gardner i K. Martain, „HRTF measurements of a KEMAR dummy-head microphone. Technical report," MIT Media Lab, 1994.

[72] J. D. Miller, M. Godfroy-Cooper i E. M. Wenzel, „Using Published HRTFS with Slab3D: Metric-Based Database Selection and Phenomena Observed," *The 20 th International Conference on Auditory Display (ICAD-2014),* 2014.

[73] J. Cronly-Dillon, K. Persaud i R. P. F. Gregory, „The perception of visual images encoded in musical form: a study in cross-modality information," *Proceedings of Biological Sciences,* tom 266, p. 2427–2433, 1999.

[74] P. Strumiłło, Elektroniczne systemy nawigacji osobistej dla niewidomych i słabowidzących, Łódź: Politechnika Łódzka, 2012.

[75] P. Zarzycki, Z. Suszyński, R. Świta, J. Łoś, M. Zarzycka i A. Kaleniecka, „Fast assessment of planar chromatographic layers quality using pulse thermovision method," *International Journal of Chromatography,* tom 1373, pp. 211-215, 2014.

Publikacje autora

[18] R. Świta, „Porównywanie pikseli obrazów kolorowych na potrzeby segmentacji," w *ICT Young 2012*, Gdańsk, 2012.

[38] R. Świta i Z. Suszyński, „Processing of thermographic sequence using Principal Component Analysis," Measurement Automation Monitoring, tom 61, nr 06, pp. 215-218, 2015.

[53] R. Świta i Z. Suszyński, „New method of segmentation of thermal image sequences," w Microelectronic Materials and Technologies tom 2, Koszalin, Wydawnictwo Politechniki Koszalińskiej, 2012, pp. 31-44.

[54] R. Świta i Z. Suszyński, „Segmentacja sekwencji obrazów metodą korelacyjną," Pomiary Automatyka Kontrola, tom 7, pp. 680-684, 2013.

[60] R. Świta i Z. Suszyński, „Cluster segmentation of thermal image sequences using kd-tree structure," International Journal of Thermophysics, tom 35, nr 12, pp. 2374-2387, 2014.

[61] R. Świta i Z. Suszyński, „Inicjalizacja segmentacji k-means uwzględniająca rozkład gęstości pikseli," Zeszyty Naukowe Wydziału Elektroniki i Informatyki Politechniki Koszalińskiej, tom 6, pp. 89-98, 2014.

[65] R. Świta i Z. Suszyński, „Hybrid watershed and region grow segmentation of color images," Zeszyty Naukowe Wydziału Elektroniki i Informatyki Politechniki Koszalińskiej, tom 3, pp. 111-122, 2011.

[75] P. Zarzycki, Z. Suszyński, R. Świta, J. Łoś, M. Zarzycka i A. Kaleniecka, „Fast assessment of planar chromatographic layers quality using pulse thermovision method," International Journal of Chromatography, tom 1373, pp. 211-215, 2014.

Koszalin, 27.10.2015

mgr inż. Robert Świta
Wydział Elektroniki i Informatyki
Politechnika Koszalińska 2015

Streszczenie pracy doktorskiej

„System wspomagania orientacji osób niewidomych oparty o konwersję obrazów 2D do postaci dźwięku przestrzennego”

Celem pracy doktorskiej było opracowanie metody konwersji obrazu do postaci dźwięku przestrzennego, ułatwiającego orientację osobie niewidomej. Transformacja obraz-dźwięk zawiera następujące etapy: przetwarzanie wstępne, segmentacja, kodowanie obrazu oraz synteza dźwięku przestrzennego. Przetwarzanie wstępne obejmuje zmianę rozdzielczości obrazu, balans bieli i filtrację statystyczną. Metoda konwersji wykorzystuje segmentację w celu wyznaczenia istotnej treści obrazu, dlatego ważnym czynnikiem wpływającym na jej jakość jest uniezależnienie od niejednorodnego oświetlenia sceny światłem białym. Zdefiniowana została przestrzeń kolorów HCI (Hue, Chroma, Intensity) i wyznaczona w niej miara odległości pomiędzy kolorami, umożliwiająca zmniejszenie wpływu nierównomierności oświetlenia. Jakość wykorzystanych algorytmów segmentacji, w tym opracowanego algorytmu segmentacji wododziałowo-rozrostowej, została porównana z wykorzystaniem obrazów referencyjnych projektu BSDS500 Uniwersytetu Berkeley.
Syntezowany dźwięk zawiera informacje o kształcie, wielkości i położeniu przestrzennym obiektu sceny. Obiekt dzielony jest na kolumny, transformowane w kolejności od lewej do prawej na tony instrumentu muzycznego. Wysokość kolumny decyduje o wysokości tonu instrumentu, natomiast środek kolumny jest interpretowany jako położenie źródła dźwięku. Liczba kolumn i czas odtwarzania dźwięku jest stała dla każdego obiektu i jest parametrem przetwarzania. Ze względu na konieczność rozróżniania położenia źródła dźwięku w pionie, do generacji dźwięku wykorzystana została funkcja transmitancji HRTF dla lewego i prawego ucha.
Równoległym torem przetwarzania jest próba kontekstowego rozpoznania obiektów wybranych scen przez układ dwóch sieci neuronowych. Dokonuje on rozpoznania obiektu na podstawie jego kształtu i tekstury. Do kodowania wejść SSN wykorzystane zostały bloki analizy metodą czynnikową PCA. Umożliwiają one uproszczenie konstruowanych sieci i przyspieszenie ich uczenia przy zachowaniu istotnej informacji wejściowej.
Z przeprowadzonych badań wynika, że rozpoznawanie kształtów i ich położenia, nawet w przypadku bardzo dobrej segmentacji, wymaga treningu. Metoda konwersji wymaga oswojenia się i zapamiętania kluczowych wzorców. Dlatego stworzony został moduł do nauki systemu. Jest on zintegrowany z edytorem grafiki, umożliwiającym łatwe konstruowanie i edycję kształtów geometrycznych oraz ich przekształcanie na dźwięk przestrzenny.
W celu potwierdzenia tezy o możliwości analizy kształtu, wielkości i położenia obiektu sceny na podstawie tak zakodowanego dźwięku przestrzennego, wykonano następujące eksperymenty:
- test rozpoznawania kształtu
- test rozpoznawania kierunku lokalizacji obiektu
- szacowanie odległości i wielkości obiektu dzięki uwzględnieniu efektu perspektywy

Na podstawie przeprowadzonych eksperymentów można stwierdzić, że dokładność lokalizacji położenia źródła dźwięku jest większa dla kierunku poziomego niż pionowego, ale również zauważalna jest różnica wrażliwości na zmianę położenia źródła w obszarach powyżej i poniżej linii horyzontu. Detekcja położenia źródła dźwięku ma bardzo istotny wpływ na możliwość analizy kształtu. Z tego powodu jednym z najbardziej istotnych kierunków rozwoju proponowanej metody powinno być doskonalenie selektywności rozdzielczości przestrzennej dźwięku, m.in. poprzez indywidualny pomiar charakterystyki HRTF.

Koszalin 27.10.2015

System for spatial orientation support of blind people, using the conversion of 2D images into a spatial sound

M.Sc Robert Świta

Koszalin University of Technology Faculty of Electronics and Computer Science

PhD DISSERTATION SUMMARY

PhD thesis is dedicated to the development of support system for spatial orientation of blind people. The work results in transformation method of images recorded by the camera into the information cues about the details of the object, concerning its shape and location. Developed algorithms are also implemented in a computer program. There exist two different mechanisms for identifying the scene - the first, major one, is a transformation based on the characteristics of the shape and position of the central subject of the scene into a spatial sound and second - auxiliary - is based on the contextual recognition of objects from selected scenes by trained ANN preceded by PCA modules.

Dissertation is divided into eight chapters, arranged in a manner consistent with the data processing order.

After the introduction, presenting an overview of existing systems of hearing images and the concept of the proposed solution, in the second chapter, the basics of color theory and color space used for comparing pixels during image segmentation is presented.

The third chapter discusses ways to interpolate images and shows the method for creating color palettes and implementation of the class *Pixelmap* that enables convenient use of the images and palettes of pseudocolors. The basic methods of automatic white balance were presented and proposed combination of their advantages in a new solution.

Chapter four begins with mathematical background for principal components analysis PCA and SVD matrix factorization. PCA method enables reduction and filtering of data by determining features that are mutually linearly independent and is widely used in the processing systems. Alongside well-known and widely used algorithms, suggestions and author's modifications of these algorithms were presented. An essential part of chapter four, however, is a description of region segmentation methods and hybrid segmentation, which is a combination of region growing and watershed segmentation methods. The result of these algorithms was compared with other types of segmentation (not necessarily working in real time) using a UC Berkeley project BSDS500.

The PCA analysis was used during automatic object recognition, after segmentation. Implementation uses two neural network modules preceded by a PCA blocks, and is described in chapter five.

Chapter six presents the developed method of sound generation based on the image of central segment. This segment is divided into columns. The height of the columns determines height of the tones of the musical MIDI instrument. Columns are processed sequentially,

synthesizing a unique melody. This way the size of the object can be presented. Existence of sound sources at the columns centers location is simulated using HRTF filtration. The shape and location of the object can be determined by an analysis or recognition of melody and virtual sound source position.

The application is also equipped with a learning module. Description of the module is a topic for chapter seven. Embedded editor allows creation of arbitrarily complex shapes thanks to the grouping and ungrouping of simple shapes. Shapes and their groups can be transformed by 2D affine transformations and converted into spatial sound.

Chapter eight is a collection of gathered conclusions resulting from the study, implementation and experiments followed by summary.

Appendix A contains a list of available instruments in the General MIDI standard.