Korekcja barwna w systemach wizji wszechogarniającej

Autor: Inż. Dominik Ługowski

Promotor: dr inż. Adrian Dziembowski

Spis treści

Wstęp

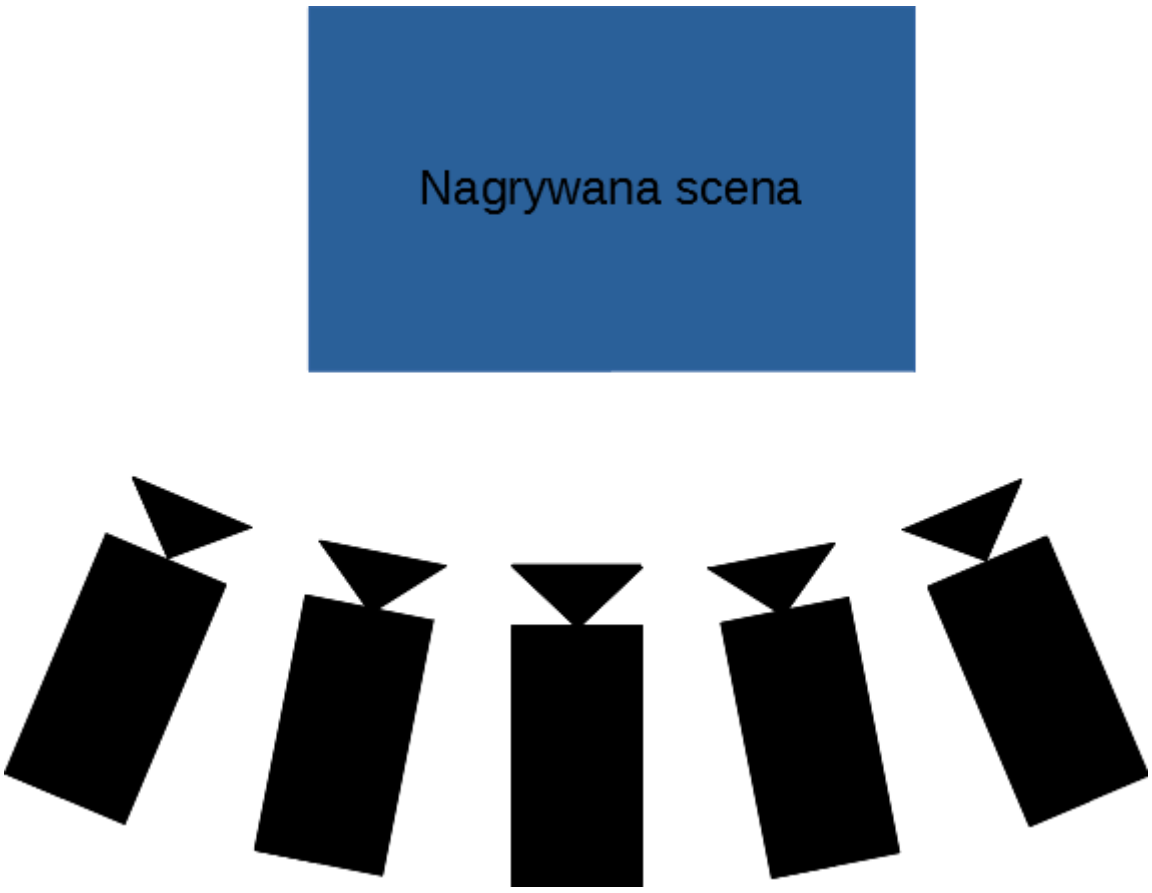
Celem niniejszej pracy było opracowanie metod korekcji barwnej dla systemów wizji wszechogarniającej oraz ocena efektywności uzyskanych rozwiązań. W związku z tym zrealizowano algorytmy korekcji koloru oraz algorytm syntezy widoków wirtualnych. Niniejsze rozwiązania zostały napisane w języku Matlab.

Systemy wizji wszechogarniającej cieszą się coraz większym zainteresowaniem w wielu branżach, np. rozrywkowej. W takich systemach widz nie jest ograniczony tylko do jednego widoku – może swobodnie przełączać się pomiędzy obrazami zarejestrowanymi przez kamery systemu oraz widokami wirtualnymi uzyskanymi w wyniku syntezy rzeczywistych widoków.

Jednym z problemów takich systemów są różnice pomiędzy charakterystykami barwnymi obrazów uzyskanych z danych kamer systemu. Wynikają one m.in. ze różnic między parametrami używanych kamer czy nierównomiernego oświetlenia nagrywanej sceny.

1. Systemy wizji wszechogarniającej

1.1. Idea



Rys. 1.1. Przykładowy system wizji wszechogarniającej

Systemy wizji wszechogarniającej, nazywane równie systemami telewizji swobodnego punktu widzenia (ang. *Free-Viewpoint Television*) czy też systemami wielokamerowymi/wielowidokowymi, to rodzaj telewizji interaktywnej, w której użytkownik ma możliwość swobodnej nawigacji w zarejestrowanej scenie [4][7]. Obecnie nie istnieje żaden komercyjny i powszechny system telewizji swobodnego punktu widzenia.

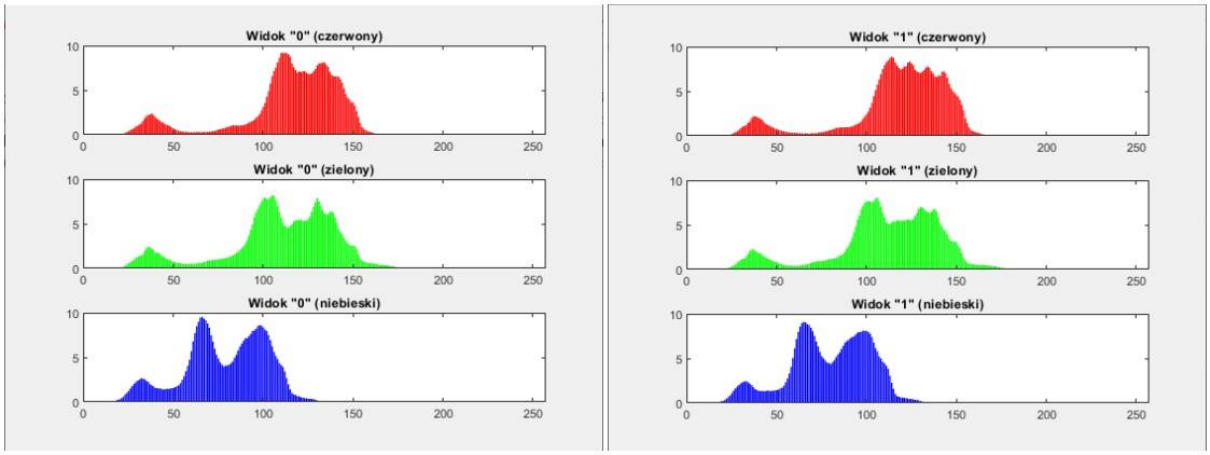
System składa się z od kilku do kilkudziesięciu kamer nagrywających daną scenę (**Rys. 1**) [4]. Takie rozwiązanie, po przetworzeniu sygnałów z kamer i złożeniu ich w jedną sekwencję, daje nowe możliwości doświadczania oglądanej sceny przez widza. W tradycyjnej telewizji to reżyser bądź realizator decyduje na czym ma być skupiona uwaga oglądających z kolei systemy wielokamerowe pozwalają na swobodną nawigację w danej scenie [10]. Możliwe jest to dzięki wykorzystaniu widoków wirtualnych uzyskanych w wyniku syntezy. Metoda syntezy widoków wirtualnych w niniejszej pracy omówiona jest w rozdziale 3.

1.2. Nierówność charakterystyk barwnych

Jak wcześniej wspomniano w systemach wielowidokowych widz może swobodnie przełączać się między nagranymi widokami.

Rys 1.2. Widoki 0 i 1 sekwencji *Ballet*

Na powyższym zdjęciu zaprezentowane są widoki z kamer „0" i „1" dla sekwencji *BT*. Możemy zaobserwować, że w związku z niedużą liczbą zmian między obrazami (widocznymi głównie przy bocznych krańcach obu obrazów) kamery są rozmieszczone stosunkowo blisko siebie. Zauważalne są również minimalne różnice jeśli chodzi o kolory.



Rys 1.3 Histogramy barwne widoków 0 i 1 dla sekwencji *Ballet*

Wspomniane różnice wynikają z nierówności charakterystyk barwnych (zwanych również charakterystykami kolorystycznymi) widocznych na histogramach na **zdjęciu 1.2.2.** Wynikają one z:

- nierównomiernego oświetlenia nagrywanej sceny, a szczególnie dla systemów z rzadkim rozmieszczeniem kamer,

- różnice pomiędzy parametrami kamer – dwie kamery tego samego modelu zawsze różną odpowiedź na to samo pobudzenie [11].

Przełączanie się pomiędzy widokami o widocznie różnych charakterystykach barwnych może powodować u widza dyskomfort. W celu zwalczania tego efektu stosuje się **korekcję barwną.**[1]

2. Korekcja barwna

2.1. Przegląd algorytmów korekcji

Głównym celem korekcji barwnej jest zrównanie charakterystyk barwnych pochodzących z różnych widoków. Algorytmy korekcji możemy podzielić na:

- parametryczne – zakładające istnienie zależności pomiędzy przestrzeniami barwnymi porównywanych obrazów,

- nieparametryczne – zakładają brak istnienia wspomnianych zależności [5].

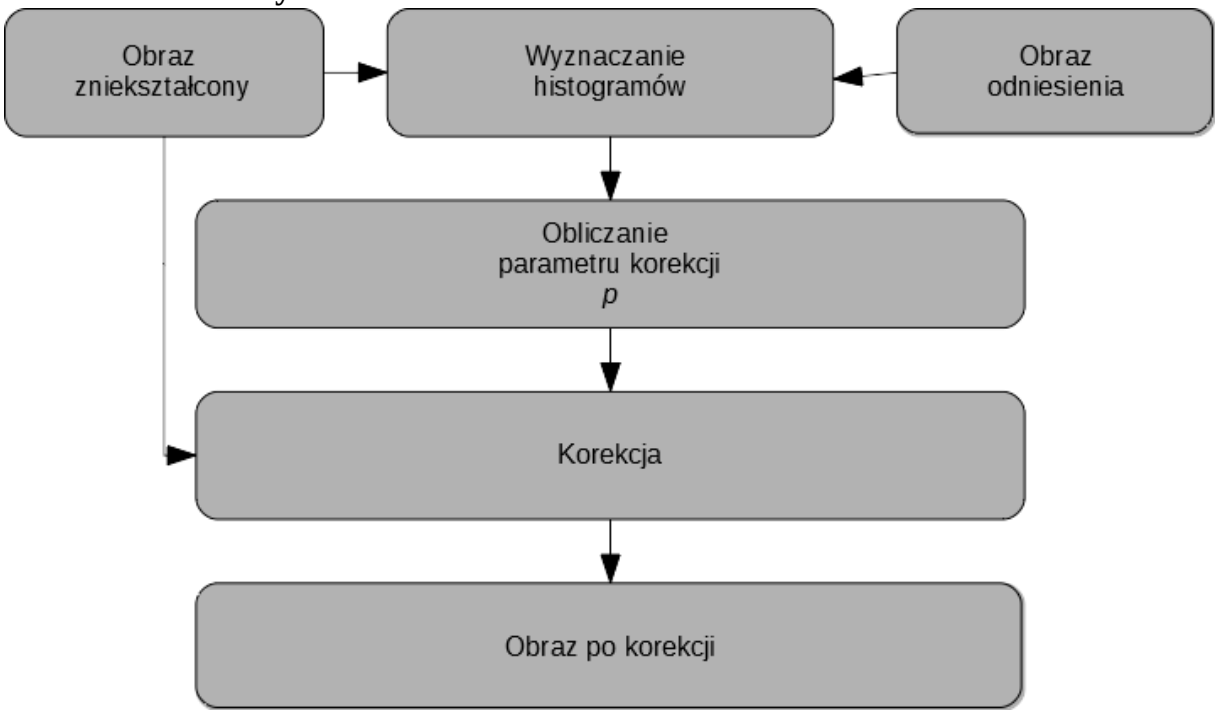
Do algorytmów parametrycznych zaliczamy algorytmy liniowy i przeniesienia koloru oraz metody wykorzystujące wzornik testowy [1]. Przykładem algorytmu nieparametrycznego jest algorytm dopasowania histogramów. W niniejszej pracy wykorzystane zostały algorytmy: liniowy [1], przeniesienia koloru [8] oraz dopasowania histogramów [9]. W korekcji rozróżniamy obrazy zniekształcony (wykorzystywany do wyznaczania parametrów korekcji, które są potem na nim stosowane) oraz odniesienia (wykorzystywany tylko do wyznaczania parametrów korekcji). Korekcja dokonywana jest osobno dla każdej składowej przestrzeni barwnej, w której reprezentowany jest obraz.

Algorytmy w niniejszej pracy są powszechnie używane w korekcji i dobrze znane, dlatego zdecydowano się na ich użycie [6]. Z powodu ograniczeń technicznych zrezygnowano z wykorzystania algorytmów wykorzystujących wzornik testowy. Przy opisywaniu zasady działania wymienionych algorytmów posłużono się kodem napisanym w języku Matlab.

2.2. Algorytm liniowy

W niniejszym algorytmie wykorzystywana jest **przestrzeń barwna RGB**.

2.2.1. Schemat blokowy



Rys 2.1 Algorytm liniowy

2.2.2. Implementacja

```
% Wyznaczanie histogramów
mainImage;
referenceImage;

histRedMain = zeros(1,256);
histGreenMain = zeros(1,256);
histBlueMain = zeros(1,256);

histRedRef = zeros(1,256);
histGreenRef = zeros(1,256);
histBlueRef = zeros(1,256);

for c=0:255
    for w=1:width
        for h=1:height
            if(mainImage(w,h,1)==c)
                histRedMain(c+1)=histRedMain(c+1)+1;
            end
            if(mainImage(w,h,2)==c)
                histGreenMain(c+1)=histGreenMain(c+1)+1;
            end
            if(mainImage(w,h,3)==c)
                histBlueMain(c+1)=histBlueMain(c+1)+1;
            end

            if(referenceImage(w,h,1)==c)
                histRedRef(c+1)=histRedRef(c+1)+1;
            end
            if(referenceImage(w,h,2)==c)
                histGreenRef(c+1)=histGreenRef(c+1)+1;
            end
            if(referenceImage(w,h,3)==c)
                histBlueRef(c+1)=histBlueRef(c+1)+1;
            end
        end
    end
end
```

Kod 2.1 Algorytm liniowy cz. 1

W pierwszym kroku wyznaczane są histogramy dla obrazów zniekształconego (*mainImage*) i odniesienia (*referenceImage*) osobno dla składowych czerwonej, zielonej i niebieskiej. Histogramy dla każdej barwy wyznaczane są według wzoru [1]:

$$hist(c) = \sum_{i=1}^{h}\sum_{j=1}^{w} \delta(c, obraz(i,j)),$$

gdzie:

$\delta(x,y) = 1,\ jeśli\ x = y,$

$\delta(x,y) = 0,\ jeśli\ x \neq y,$

hist – histogram dla danej składowej barwnej,

c – wartość próbki dla danej barwy,

h, w – wysokość i szerokość,

i, j – współrzędne w obrazie.

```
% Obliczenie parametru korekcji
sumMainHistRed = 0;
sumMainHistBlue = 0;
sumMainHistGreen = 0;
sumRefHistRed = 0;
sumRefHistBlue = 0;
sumRefHistGreen = 0;

for v=1:256
    sumMainHistRed = sumMainHistRed + histRedMain(v)*v;
    sumMainHistGreen = sumMainHistGreen + histGreenMain(v)*v;
    sumMainHistBlue = sumMainHistBlue + histBlueMain(v)*v;

    sumRefHistRed = sumRefHistRed + histRedRef(v)*v;
    sumRefHistGreen = sumRefHistGreen + histGreenRef(v)*v;
    sumRefHistBlue = sumRefHistBlue + histBlueRef(v)*v;
end
```

Kod 2.2 Algorytm liniowy cz. 2

W następnym kroku niezależnie dla każdej składowej obliczany jest parametr korekcji *p [1]*:

$$p = \frac{\sum_c histMain(c)\ \cdot\ c}{\sum_c histRef\ (c) \cdot c},$$

gdzie:

histMain – histogram obrazu zniekształconego (np. *histRedMain*),

histRef – histogram obrazu odniesienia (np. *histRedRef),*

c – wartość próbki dla danej barwy,

sumMainHistRed – po wykonaniu pętli to dla składowej czerwonej obrazu zniekształconego wartość równania:

$$\sum_{c} histMainRed(c) \cdot c.$$

Analogicznie otrzymujemy wartości dla pozostałych składowych i obrazu odniesienia.

correctionRed, *correctionGreen* i *correctionBlue* to wartości parametru *p* kolejno dla składowych czerwonej, zielonej i niebieskiej.

```
% Przetwarzanie obrazu
mainImage(:,:,1) = mainImage(:,:,1)*correctionRed;
mainImage(:,:,2) = mainImage(:,:,2)*correctionGreen;
mainImage(:,:,3) = mainImage(:,:,3)*correctionBlue;
```

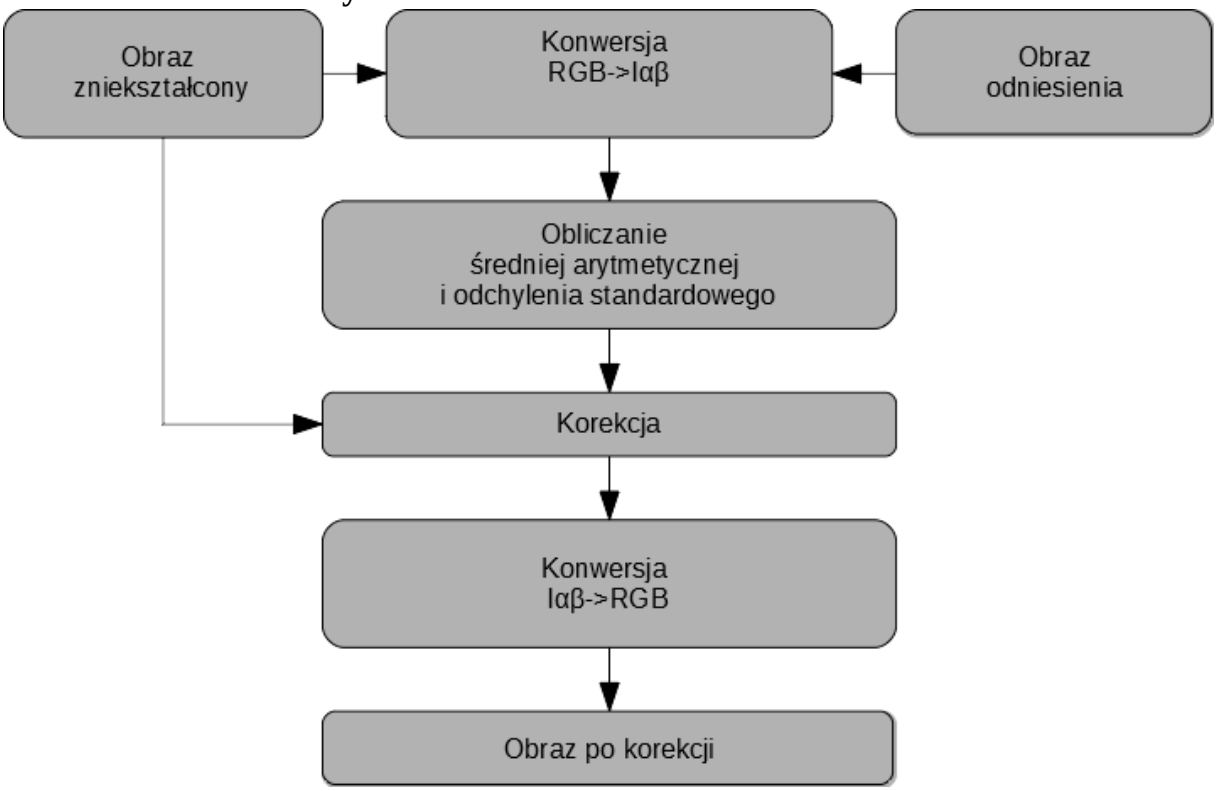
Kod 2.3 Algorytm linowy cz. 3

W ostatnim kroku każdy punkt obrazu zniekształconego mnożony jest przez współczynnik korekcji *p* tworząc w ten sposób obraz skorelowany.

2.3. Algorytm przeniesienia koloru

W niniejszym algorytmie, w związku z niską korelację między składowymi, wykorzystano **przestrzeń barwną lαβ** wyznaczaną z przestrzeni RGB [8].

2.3.1. Schemat blokowy



Rys. 2.2 Algorytm przeniesienia koloru

2.3.2. Konwersja do przestrzeni lαβ

W celu otrzymania wartości lαβ należy wykorzystać przestrzeń LMS otrzymywaną z przestrzeni RGB w następujący sposób [8]:

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.3811 & 0.5783 & -0.0402 \\ 0.1967 & 0.7244 & 0.0782 \\ 0.0241 & 0.1228 & 0.8444 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

Przekształcenie odwrotne uzyskuje się za pomocą następującego równania:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 4.4679 & -3.5873 & 0.1193 \\ -1.2186 & 2.3809 & -0.1624 \\ 0.0497 & -0.2439 & 1.2045 \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix}.$$

Z kolei przekształcenie do przestrzeni lαβ wykonywane jest w następujący sposób:

$$\begin{bmatrix} l \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{3}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{6}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -2 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} L' \\ M' \\ S' \end{bmatrix},$$

gdzie:

$$L' = log_{10}L$$
$$M' = log_{10}M.$$
$$S' = log_{10}S$$

Powrót do przestrzeni L'M'S':

$$\begin{bmatrix} L' \\ M' \\ S' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -2 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{3}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{6}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} l \\ \alpha \\ \beta \end{bmatrix}.$$

2.3.3. Implementacja

```
%Zmiana z przestrzeni RGB na l_alpha_beta
labMainImage = rgb2lab(mainImage);
labRefImage = rgb2lab(referenceImage);
```

Kod 2.4 Przeniesienie koloru cz.1

W języku Matlab przejście z przestrzeni RGB do lαβ realizowane jest za pomocą jednej funkcji *rgb2lab* podobnie jak powrót do RGB przy użyciu *lab2rgb*.

```
%Obliczenie średniej i odchylenia standardowego dla każdej składowej w
%obu obrazach

% Obraz główny
% matrixToArray służy wyłącznie do zmiany obrazu w wektor, aby móc
% poprawnie zastosować funkcję mean()
matrixToArray = labMainImage(:,:,1);
matrixToArray = matrixToArray(:);
meanML = mean(matrixToArray);
stdML = std(matrixToArray);

matrixToArray = labMainImage(:,:,2);
matrixToArray = matrixToArray(:);
meanMA = mean(matrixToArray);
stdMA = std(matrixToArray);

matrixToArray = labMainImage(:,:,3);
matrixToArray = matrixToArray(:);
meanMB = mean(matrixToArray);
stdMB = std(matrixToArray);

% Obraz odniesienia
matrixToArray = labRefImage(:,:,1);
matrixToArray = matrixToArray(:);
meanRefL = mean(matrixToArray);
stdRefL = std(matrixToArray);

matrixToArray = labRefImage(:,:,2);
matrixToArray = matrixToArray(:);
meanRefA = mean(matrixToArray);
stdRefA = std(matrixToArray);

matrixToArray = labRefImage(:,:,3);
matrixToArray = matrixToArray(:);
meanRefB = mean(matrixToArray);
stdRefB = std(matrixToArray);
```

Kod 2.5 Przeniesienie koloru cz. 2

Następnym krokiem algorytmu jest obliczenie średniej arytmetycznej (np. *meanML*) i odchylenia standardowego (np. *stdRefA*) dla każdej składowej (wykonywane odpowiednio przy pomocy funkcji *mean* oraz *std*).

```
% Uzyskanie obrazu zniekształconego
labMainImage(:,:,1) = (labMainImage(:,:,1) - meanML) * (stdML/stdRefL) + meanRefL;
labMainImage(:,:,2) = (labMainImage(:,:,2) - meanMA) * (stdMA/stdRefA) + meanRefA;
labMainImage(:,:,3) = (labMainImage(:,:,3) - meanMB) * (stdMB/stdRefB) + meanRefB;

% l_alpha_beta -> RGB + wyświetlenie
newImage = lab2rgb(labMainImage,'OutputType','uint8');
```

Kod 2.6 Przeniesienie koloru cz. 3

Końcowy etap to podstawienie uzyskanych średnich i odchyleń standardowych do równania:

$$p' = (p_z - \mu_z) \cdot {}^{\sigma_z}/_{\sigma_o} + \mu_o,$$

gdzie:

p – wartość punktu obrazu zniekształconego dla wybranej składowej l, α lub β,

p' – nowa wartość punktu obrazu zniekształconego dla wybranej składowej l, α lub β,

μ_z – średnia z wartości punktów obrazu odniesienia dla wybranej składowej l, α lub β (np. *meanMB*),

μ_o – średnia z wartości punktów obrazu odniesienia dla wybranej składowej l, α lub β (np. *meanRefL*),

σ_z – odchylenie standardowe z obrazu zniekształconego dla wybranej składowej l, α lub β (np. *stdMA*),

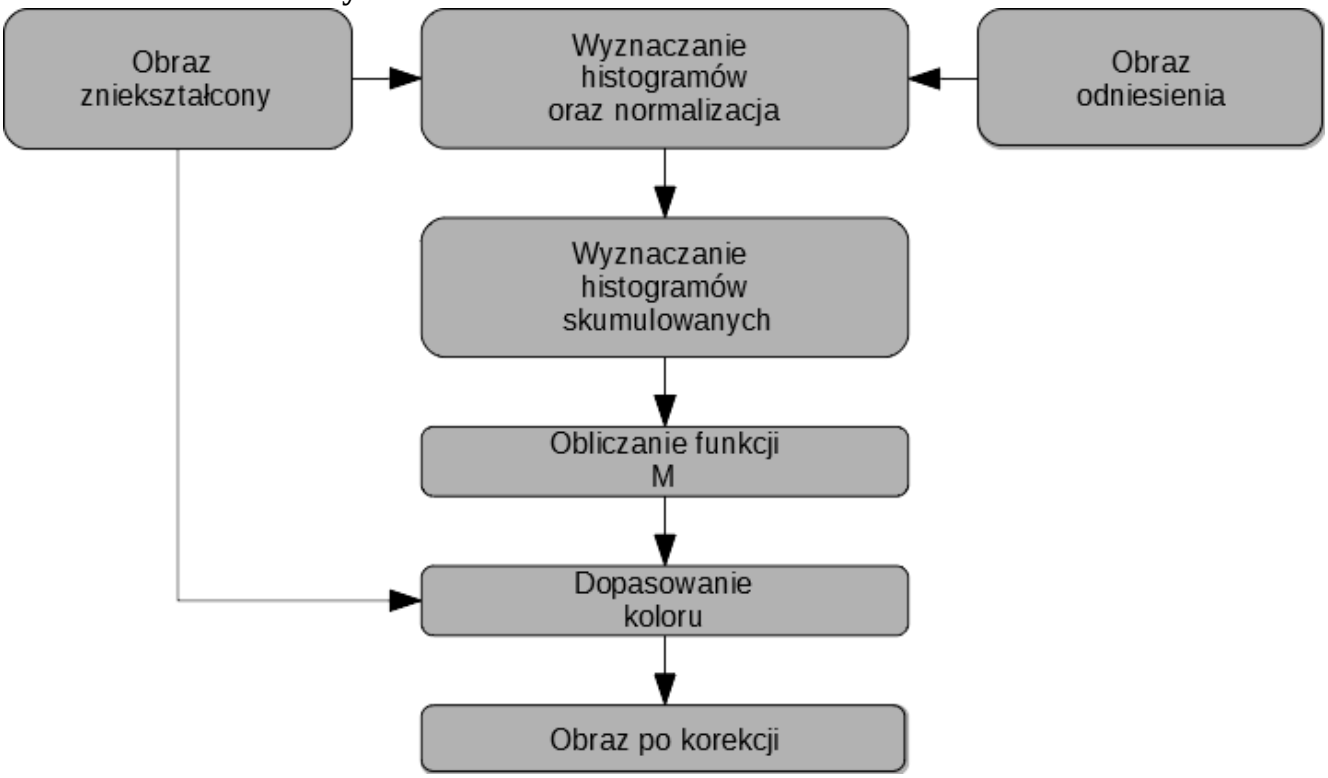
σ_o – odchylenie standardowe z obrazu zniekształconego dla wybranej składowej l, α lub β (np. *stdRefA*).

Po wyznaczeniu nowych wartości obrazu dokonywana jest konwersja z lαβ na RGB (*newImage*).

2.4. Algorytm dopasowania histogramów

W niniejszym algorytmie wykorzystywana jest **przestrzeń barwna RGB**.

2.4.1. Schemat blokowy



Schemat 2.4.1. Dopasowanie histogramów

2.4.2. Implementacja

```
%Tworzenie histogram
for c=0:255
    for w=1:height
        for h=1:width
            %Original image
            if(mainImage(w,h,1)==c)
                histRed(c+1)=histRed(c+1)+1;
            end
            if(mainImage(w,h,2)==c)
                histGreen(c+1)=histGreen(c+1)+1;
            end
            if(mainImage(w,h,3)==c)
                histBlue(c+1)=histBlue(c+1)+1;
            end
            if(referenceImage(w,h,1)==c)
                histRedRef(c+1)=histRedRef(c+1)+1;
            end
            if(referenceImage(w,h,2)==c)
                histGreenRef(c+1)=histGreenRef(c+1)+1;
            end
            if(referenceImage(w,h,3)==c)
                histBlueRef(c+1)=histBlueRef(c+1)+1;
            end
        end
    end
    %Normalizacja
    histRed(c+1)=histRed(c+1)/(width+height);
    histGreen(c+1)=histGreen(c+1)/(width+height);
    histBlue(c+1)=histBlue(c+1)/(width+height);

    histRedRef(c+1)=histRedRef(c+1)/(width+height);
    histGreenRef(c+1)=histGreenRef(c+1)/(width+height);
    histBlueRef(c+1)=histBlueRef(c+1)/(width+height);

end
```

Kod 2.7 Dopasowanie histogramów cz. 1

Pierwszym krokiem jest wyznaczenie histogramów (np. *histRed*, *histRedRef*) dla obrazu zniekształconego (*mainImage*) oraz obrazu odniesienia (*referenceImage*) i normalizacja uzyskanych wartości. Proces ten można opisać wzorem [1]:

$$hist(c) = \frac{1}{h * w} \sum_{i=1}^{h} \sum_{j=1}^{w} \delta(c, obraz(i,j)),$$

gdzie:

$\delta(x,y) = 1, \ jeśli \ x = y,$

$\delta(x,y) = 0, \ jeśli \ x \neq y,$

c – wartość próbki dla danej barwy,

hist – histogram dla danej składowej barwnej,

h, w – wysokość i szerokość,

i, j – współrzędne punktu w obrazie.

```
%Wyznaczanie histogramu skumulowanego
CummulatedMainRed = zeros(1,256);
CummulatedMainBlue = zeros(1,256);
CummulatedMainGreen = zeros(1,256);

sumRed = 0;
sumGreen = 0;
sumBlue = 0;

for v=1:256
    for i=1:v
        sumRed = sumRed+histRed(i);
        sumGreen = sumGreen+histGreen(i);
        sumBlue = sumBlue+histBlue(i);
    end
    CummulatedMainRed(v) = sumRed;
    CummulatedMainGreen(v) = sumGreen;
    CummulatedMainBlue(v) = sumBlue;
    sumRed = 0;
    sumGreen = 0;
    sumBlue = 0;
end
```

Kod 2.8 Dopasowanie histogramów cz. 2

Następny etap to wyznaczenie histogramu skumulowanego (*CummulatedMain* dla obrazu zniekształconego i *CummulatedRef* dla obrazu odniesienia) dla każdej składowej barwnej:

$$CummulatedHistogram(c) = \sum_{i=0}^{c} hist(i),$$

gdzie:

c – wartość próbki dla danej barwy,

hist(i) – liczba próbek w obrazie o wartości *i*,

Cummulated(c) – histogram skumulowany.

```
%Obliczanie funkcji M
Mred = zeros(1,256);
Mgreen = zeros(1,256);
Mblue = zeros(1,256);

for v=1:256
    for u=1:256
        if(CummulatedMainRed(v) >=CummulatedRefRed(u))
            if(u==256)
                Mred(v)=u;
                break;
            end
            if(CummulatedMainRed(v)<CummulatedRefRed(u+1))
                Mred(v)=u;
                break;
            end
        end
    end
    for u=1:256
        if(CummulatedMainGreen(v) >=CummulatedRefGreen(u))
            if(u==256)
                Mgreen(v)=u;
                break;
            end
            if(CummulatedMainGreen(v)<CummulatedRefGreen(u+1))
                Mgreen(v)=u;
                break;
            end
        end
    end
    for u=1:256
        if(CummulatedMainBlue(v) >=CummulatedRefBlue(u))
            if(u==256)
                Mblue(v)=u;
                break;
            end
            if(CummulatedMainBlue(v)<CummulatedRefBlue(u+1))
                Mblue(v)=u;
                break;
            end
        end
    end
end
```

Kod 2.9 Dopasowanie histogramów cz. 3

Następny etap polega na utworzeniu funkcji przeniesienia M (np. *Mblue*) w oparciu o skumulowanego histogramy. Ów funkcję można opisać równanie [1]:

$$M(c) = u, \;\; CummulatedRef(u) \leq CummulatedMain(c) < CummulatedRef(u+1).$$

```
%Dopasowanie koloru
for w=1:width
    for h=1:height
        newImage(h,w,1) = uint8(Mred(mainImage(w,h,1)+1));
        newImage(h,w,2) = uint8(Mgreen(mainImage(w,h,2)+1));
        newImage(h,w,3) = uint8(Mblue(mainImage(w,h,3)+1));
    end
end
```

Kod 2.10 Dopasowanie histogramów cz. 4

Ostatnim krokiem jest zmiana wartości w obrazie zniekształconym na bazie funkcji przeniesienia:

$$p_d(m,n) = M\big(p_r(m,n)\big),$$

gdzie:

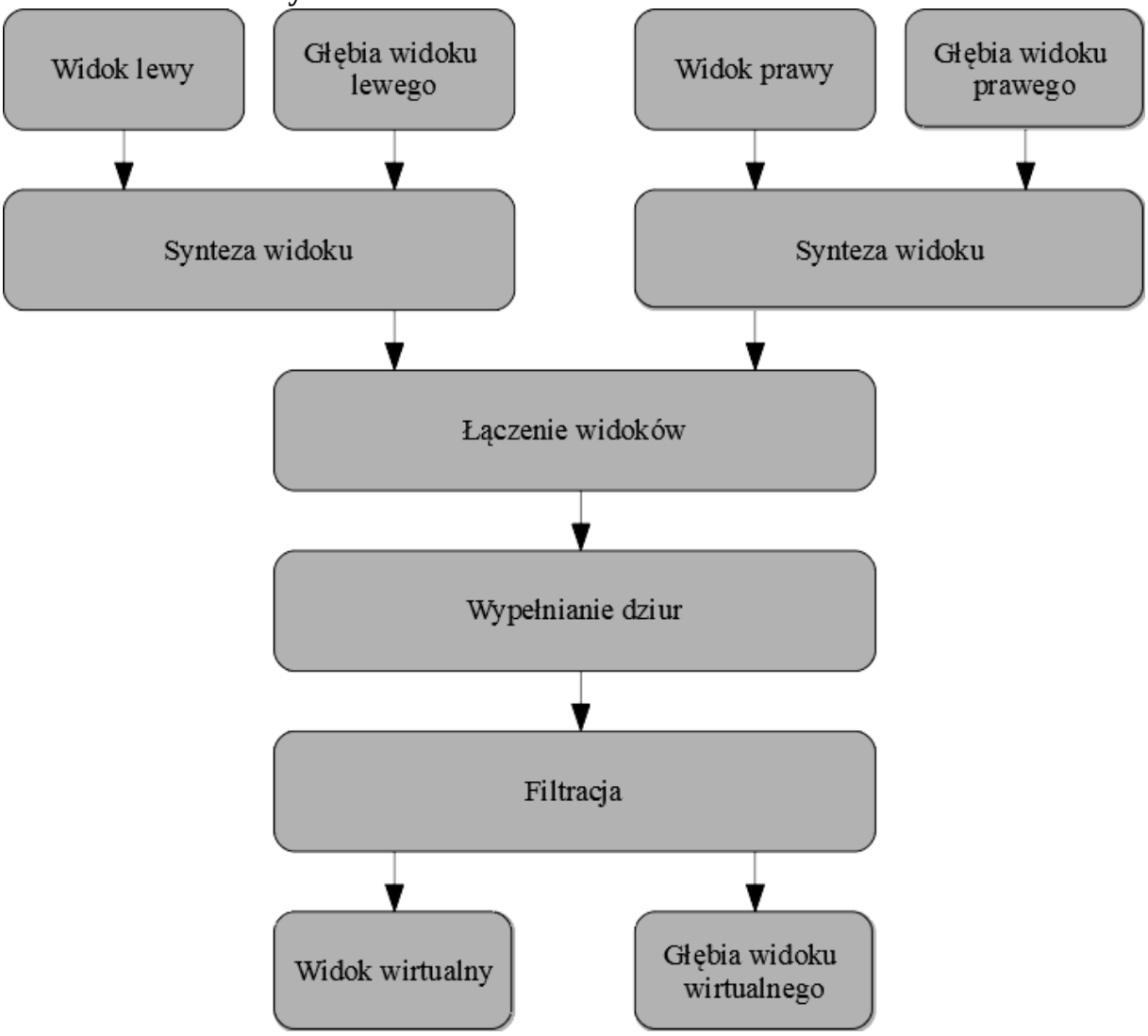
m, n – współrzędne wybranego punktu w obrazie zniekształconym,

$p_d(m,n)$ – nowa wartość w obrazie zniekształconym w punkcie o współrzędnych *(m,n),*

$p_r(m,n)$ – wartość w obrazie odniesienia w punkcie o współrzędnych *(m,n).*

3. Synteza widoków wirtualnych

3.1. Schemat blokowy



Rys. 3.1. Synteza widoków wirtualnych

3.2. Synteza widoków

3.2.1. Idea

Synteza polega na tworzeniu widoków wirtualnych na podstawie widoków rzeczywistych, informacji przestrzennych (rozmieszczeniu obiektów w scenie) oraz systemie kamer, czyli ich położeniu. Kamery opisywane są za pomocą **macierzy parametrów wewnętrznych** oraz **macierzy parametrów zewnętrznych** [2].

Macierz parametrów wewnętrznych K [4]:

$$K = \begin{bmatrix} f \cdot s_x & f \cdot s_k & o_x \\ 0 & f \cdot s_y & o_y \\ 0 & 0 & 1 \end{bmatrix},$$

gdzie:

f – ogniskowa kamery,

s_x, s_y – okresy próbkowania w kierunku poziomym i pionowym,

s_k – skośność przetwornika kamery,

o_x, o_y – położenie punktu zasadniczego.

Macierz parametrów zewnętrznych RT otrzymywana jest z **macierzy rotacji** R oraz **wektora translacji** T:

$$RT = [R_{3x3}|T_{3x1}].$$

Na podstawie tych parametrów wyznaczane są **macierze projekcji kamery rzeczywistej i wirtualnej** P_R i P_V.

$$P = \begin{bmatrix} K_{3x3} & 0_{3x1} \\ 0_{1x3} & 1 \end{bmatrix} \cdot \begin{bmatrix} R_{3x3} & T_{3x1} \\ 0_{1x3} & 1 \end{bmatrix},$$

gdzie:

K – macierz parametrów wewnętrznych,

R – macierz rotacji,

T – wektor translacji.

Do przeprowadzenia syntezy niezbędne są również informacje o położeniu obiektów. Najpopularniejszą metodą ich reprezentacji są **mapy głębi.**

Rys. 3.2 Przykładowa mapa głębi

Na podstawie tych wszystkich parametrów można dokonać rzutowania punktów z obrazu rzeczywistego na obraz wirtualny.

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = P_v \cdot {P_R}^{-1} \cdot \begin{bmatrix} x_R \\ y_R \\ z_R \\ 1 \end{bmatrix},$$

gdzie:

x_R, y_R – współrzędne danego punktu w widoku rzeczywistym,

z_R – wartość głębi w danym punkcie,

P_V, P_R – macierze projekcji wirtualna i rzeczywista.

Na podstawie x', y', z', w' wyznaczane są współrzędne punktu w widoku wirtualnym (x_v i y_v) oraz jego głębia (z_v):

$$x_v = \frac{x'}{w'}, \qquad y_v = \frac{y'}{w'}, \qquad z_v = \frac{z'}{w'}.$$

W przypadku, gdy 2 punkty z widoku rzeczywistego zostaną zrzutowane na te same współrzędne w widoku wirtualnym. W sytuacji, gdy na daną pozycję zrzutowany jest już

jeden punkt, a chcemy wpisać tam inny to porównujemy wartości głębi tych punktów. Jeżeli drugi punkt będzie miał większą wartość to nadpisuje on punkt pierwszy. W przeciwnym razie punkt drugi jest odrzucany.

Rys. 3.3 Od lewej: widok z kamery 0, widok z kamery 0 przerzutowany do kamery 1, widok rzeczywisty kamery 1

3.2.2. Implementacja

W celu uzyskania lepszej jakości widoków wirtualnych zastosowano **algorytm usuwania krawędzi**. Sprawia on, że krawędzie obiektów z widoku wirtualnego nie są brane pod uwagę w syntezie. Pozwala to na usunięcie „krawędzi-duchów" w syntezowanych obrazach.

```
numberOfCheckedNeighbours = edgeSize * 2;
margin = edgeSize + numberOfCheckedNeighbours;
```

Kod 3.1 Algorytm usuwania krawędzi cz. 1

W pierwszy kroku obliczane są liczba sprawdzanych punktów sąsiednich (*numberOfCheckedNeighbours*) oraz margines (*margin*) na podstawie zadanych podanej wielkości krawędzi (*edgeSize*).

```
for h = (1+margin):(height-margin)
    for w = (1+margin):(width-margin)
        for e = 1:edgeSize
            if(depth(h,w) < depth(h,w-e) - depthThresh)  %lewo
                diffY = 0;diffU = 0;diffV = 0;
                for n = 1:numberOfCheckedNeighbours
                    diffY = diffY + abs(Y(h,w+n) - Y(h,w+n+1));
                    diffU = diffU + abs(U(h,w+n) - U(h,w+n+1));
                    diffV = diffV + abs(V(h,w+n) - V(h,w+n+1));
                end
                anY = abs(Y(h,w) - Y(h,w+1)) * numberOfCheckedNeighbours * 5;
                anU = abs(U(h,w) - U(h,w+1)) * numberOfCheckedNeighbours * 5;
                anV = abs(V(h,w) - V(h,w+1)) * numberOfCheckedNeighbours * 5;
                if(anY > diffY || anU > diffU || anV > diffV)
                    removeMatrix(h,w) = 0;
                end
            end
            if(depth(h,w) < depth(h,w-e) - depthThresh)  %prawo
                diffY = 0;diffU = 0;diffV = 0;
                for n = 1:numberOfCheckedNeighbours
                    diffY = diffY + abs(Y(h,w-n) - Y(h,w-n-1));
                    diffU = diffU + abs(U(h,w-n) - U(h,w-n-1));
                    diffV = diffV + abs(V(h,w-n) - V(h,w-n-1));
                end
                anY = abs(Y(h,w) - Y(h,w-1)) * numberOfCheckedNeighbours * 5;
                anU = abs(U(h,w) - U(h,w-1)) * numberOfCheckedNeighbours * 5;
                anV = abs(V(h,w) - V(h,w-1)) * numberOfCheckedNeighbours * 5;
                if(anY > diffY || anU > diffU || anV > diffV)
                    removeMatrix(h,w) = 0;
                end
            end
            if(depth(h,w) < depth(h+1,w) - depthThresh)  %dół
                diffY = 0;diffU = 0;diffV = 0;
                for n = 1:numberOfCheckedNeighbours
                    diffY = diffY + abs(Y(h+n,w) - Y(h+n+1,w));
                    diffU = diffU + abs(U(h+n,w) - U(h+n+1,w));
                    diffV = diffV + abs(V(h+n,w) - V(h+n+1,w));
                end
                anY = abs(Y(h,w) - Y(h+1,w)) * numberOfCheckedNeighbours * 5;
                anU = abs(U(h,w) - U(h+1,w)) * numberOfCheckedNeighbours * 5;
                anV = abs(V(h,w) - V(h+1,w)) * numberOfCheckedNeighbours * 5;
                if(anY > diffY || anU > diffU || anV > diffV)
                    removeMatrix(h,w) = 0;
                end
            end
            if(depth(h,w) < depth(h+1,w) - depthThresh)  %góra
                diffY = 0;diffU = 0;diffV = 0;
                for n = 1:numberOfCheckedNeighbours
                    diffY = diffY + abs(Y(h-n,w) - Y(h-n-1,w));
                    diffU = diffU + abs(U(h-n,w) - U(h-n-1,w));
                    diffV = diffV + abs(V(h-n,w) - V(h-n-1,w));
                end
                anY = abs(Y(h,w) - Y(h-1,w)) * numberOfCheckedNeighbours * 5;
                anU = abs(U(h,w) - U(h-1,w)) * numberOfCheckedNeighbours * 5;
                anV = abs(V(h,w) - V(h-1,w)) * numberOfCheckedNeighbours * 5;
                if(anY > diffY || anU > diffU || anV > diffV)
                    removeMatrix(h,w) = 0;
                end
            end
        end
    end
end
```

Kod 3.2 Algorytm usuwania krawędzi cz. 2

W następnym kroku dla każdego danego punktu jest sprawdzana różnica głębi między nim a danym sąsiadem. Jeśli mieści się w zadanym zakresie (wartość bezwzględna różnicy mniejsza niż *depthThresh*) obliczane są parametry *diff* dla luminancji i chrominancji, czyli suma różnic pomiędzy danym punktem a punktami obok. Kolejno obliczany jest parametr *an*, czyli różnica między punktem a jego sąsiadem, która jest potem pomnożona razy 5 i razy liczba sprawdzanych sąsiadów. Jeśli różnica *an* jest większe od *diff* (dla luminancji bądź chrominancji) to dany punkt nie jest brany pod uwagę w syntezie. Algorytm zwraca tablicę zer i jedynek, gdzie:

- 0 – punkt o danych współrzędnych nie jest brany pod uwagę w syntezie,

- 1 – punkt o danych współrzędnych jest brany pod uwagę w syntezie.

```
%Macierze projekcji
Pr = Kr*RTr;
Pr_inverse = inv(Pr);
Pv = Kv*RTv;
P = Pv * Pr_inverse;


% Obliczenie wartości głebi Zr wg wzoru 1/Zr = Z/255 * Z'near + 1/Zfar
Znear = 42;
Zfar = 130;
invZnearMinusInvZfar = 1/Znear - 1/Zfar;

Zr = zeros(768,1024);

for h=1:height
    for w=1:width
        Z = double(Y_cam0_depth(h,w,1));
        Zr(h,w) = Z/65535 * invZnearMinusInvZfar + 1/Zfar;
        Zr(h,w) = (1/Zr(h,w));
    end
end
```

Kod 3.3 Synteza widoku wirtualnego cz. 1

W pierwszym kroku syntezy obliczane są macierze projekcji. Potem na podstawie zadanych wartości (*Znear* i *Zfar*) obliczane są wartości głębi dla każdego punktu.

```
% Synteza widoku wirtualnego kamery
xVirtual = zeros(768,1024);
yVirtual = zeros(768,1024);
zVirtual = zeros(768,1024);

Znew = zeros(768,1024);


    % val = [x'; y'; z'; w']
for h=1:height
   for w=1:width
       val = Pv* Pr_inverse * [w*Zr(h,w);h*Zr(h,w);Zr(h,w);1];
       invZ = 1 / val(3);
       xVirtualal = round(val(1) * invZ);
       yVirtualal = round(val(2) * invZ);
       if(xVirtualal > 0 && xVirtualal < width && yVirtualal > 0 && yVirtualal < height)
           %[xVirtualal yVirtualal invZ]
           if(zVirtual(yVirtualal,xVirtualal) < invZ)
               xVirtual(h,w) = xVirtualal;
               yVirtual(h,w) = yVirtualal;
               zVirtual(yVirtualal,xVirtualal) = invZ;


               Zconvert = double(double(invZ-1/Zfar)/double(invZnearMinusInvZfar));
               Zconvert = uint16(Zconvert * 65535);
               Znew(yVirtualal,xVirtualal)= Zconvert;

           end
       end
   end
end
```

Kod 3.4 Synteza widoku wirtualnego cz. 2

Kolejno obliczane są współrzędne w obrazie wirtualnym dla każdego punktu (zakładając, że na podstawie macierzy usuwania krawędzi dany punkt jest brany pod uwagę w syntezie). Dany punkt jest zapisany w macierzach rzutowania (*xVirtual* i *yVirtual*):

- mieści się w granicach obrazu (nie wychodzi poza wysokość i szerokość obrazu),

- inny punkt zrzutowany na dane współrzędne ma mniejszą wartość głębi.

```
% Rzutowanie kolorów na obraz wirtualny
for h=1:height
   for w=1:width
       x = xVirtual(h,w);
       y = yVirtual(h,w);
       if(x == 0 || y == 0)

       else
       imageVirtual(y,x,1) = imageRealCam0(h,w,1,1);
       imageVirtual(y,x,2) = imageRealCam0(h,w,2,1);
       imageVirtual(y,x,3) = imageRealCam0(h,w,3,1);

       end
   end
end
```

Kod 3.5 Synteza widoku wirtualnego cz. 3

W ostatnim kroku rzutowane są punkty z widoku rzeczywistego na współrzędne wskazane w macierzach rzutowania.

3.3. Łączenie widoków

3.3.1. Idea

W celu uzyskania widoku wirtualnego środkowego należy połączyć widoki wirtualne z uzyskane z kamer lewej i prawej [2]. Algorytm przechodzi punkt po punkcie i porównuje wartości głębi w dwóch widokach. Punkt widoku łączonego:

- przyjmuje wartość punktu o większej wartości głębi (punktu będącego bliżej),

- wpisywana jest średnia wartości z obydwu widoków jeśli wartości ich głębi są równe.

Rys. 3.4 Od lewej: widok wirtualny stworzony na podstawie informacji przerzutowanej z kamery lewej, widok stworzony na podstawie informacji przerzutowanej z kamery prawej, połączone widoki

3.3.2. Implementacja

```
v0 = double(view0);
v1 = double(view1);
for w=1:width
    for h=1:height
        if(depth0(h,w)>depth1(h,w))
            mergedView(h,w,1)=v0(h,w,1);
            mergedView(h,w,2)=v0(h,w,2);
            mergedView(h,w,3)=v0(h,w,3);
            mergedDepth(h,w)=depth0(h,w);
        elseif(depth0(h,w)<depth1(h,w))
            mergedView(h,w,1)=v1(h,w,1);
            mergedView(h,w,2)=v1(h,w,2);
            mergedView(h,w,3)=v1(h,w,3);
            mergedDepth(h,w)=depth1(h,w);
        else
            mergedView(h,w,1)=(v0(h,w,1)/2) + (v1(h,w,1)/2);
            mergedView(h,w,2)=(v0(h,w,2)/2) + (v1(h,w,2)/2);
            mergedView(h,w,3)=(v0(h,w,3)/2) + (v1(h,w,3)/2);
            mergedDepth(h,w)=depth1(h,w);
        end
    end
end
```

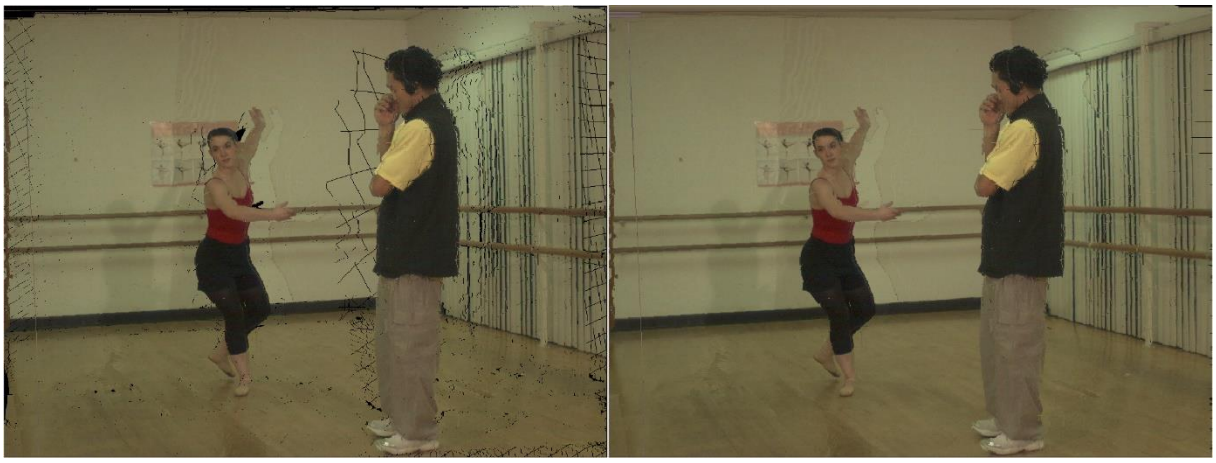
Kod 3.6 Łączenie widoków

Algorytm przyrównuje wartości głębi na danych współrzędnych (*depth0* oraz *depth1*). Punkt widoku połączonego (*mergedView*) i głębi połączonej (*mergedDepth*) przyjmuje wartość z punktu o większej głębi, a jeśli wartości głębi są równo to wpisywana jest średnia z obu obrazów.

3.4. Wypełnianie dziur

3.4.1. Idea

Po połączeniu dwóch widoków na obrazie dalej mogą występować dziury, czyli punkty o zerowej wartości wszystkich składowych barwnych. Pojawiają się one, gdy na dane współrzędne nie został zrzutowany żaden punkt z obu widoków. Wypełniane są one poprzez uśrednienie dwóch najbliższych nie-dziur (szukanych w poziomie) [2].



Rys. 3.5 Od lewej: widok wirtualny przed i po wypełnieniu dziur

3.4.2. Implementacja

```
for h = 1:height
   for w = 1:width
      r = filledImage(h,w,1);
      g = filledImage(h,w,2);
      b = filledImage(h,w,3);
      if(r == 0 && g == 0 && b == 0)
          if(w == 1)
              position = searchForNonEmpty(filledImage, w, h, width, "right");
              filledImage(h,w,1) = filledImage(h,position,1);
              filledImage(h,w,2) = filledImage(h,position,2);
              filledImage(h,w,3) = filledImage(h,position,3);
              filledDepth(h,w) = filledDepth(h,position);
          elseif(w == width)
              position = searchForNonEmpty(filledImage, w, h, width, "left");
              filledImage(h,w,1) = filledImage(h,position,1);
              filledImage(h,w,2) = filledImage(h,position,2);
              filledImage(h,w,3) = filledImage(h,position,3);
              filledDepth(h,w) = filledDepth(h,position);
          else
              positionR = searchForNonEmpty(filledImage, w, h, width, "right");
              positionL = searchForNonEmpty(filledImage, w, h, width, "left");
              filledImage(h,w,1) = uint8(mean([filledImage(h,positionL,1),filledImage(h,positionR,1)]));
              filledImage(h,w,2) = uint8(mean([filledImage(h,positionL,2),filledImage(h,positionR,2)]));
              filledImage(h,w,3) = uint8(mean([filledImage(h,positionL,3),filledImage(h,positionR,3)]));
              filledDepth(h,w) = uint16(mean([filledDepth(h,positionL),filledDepth(h,positionR)]));
          end
      end
   end
end
```

Kod 3.7 Wypełnianie dziur

W pętli głównej programu wyszukiwane są dziury do wypełnienia. Jeśli wartości składowych czerwonej, zielonej i niebieskiej są równe 0 to dany punkt jest sklasyfikowany jako dziura. Wtedy wyszukiwane są dwie najbliższe nie dziury (*searchForNonEmpty*) z prawej i lewej strony, a następnie obliczana z nich średnia

(zarówno dla widoku rzeczywistego jak i głębi) i wpisywana w miejsce dziury. W przypadku, gdy operujemy na lewej lub prawej krawędzi widoku to wyszukiwany jest tylko jeden punkt.

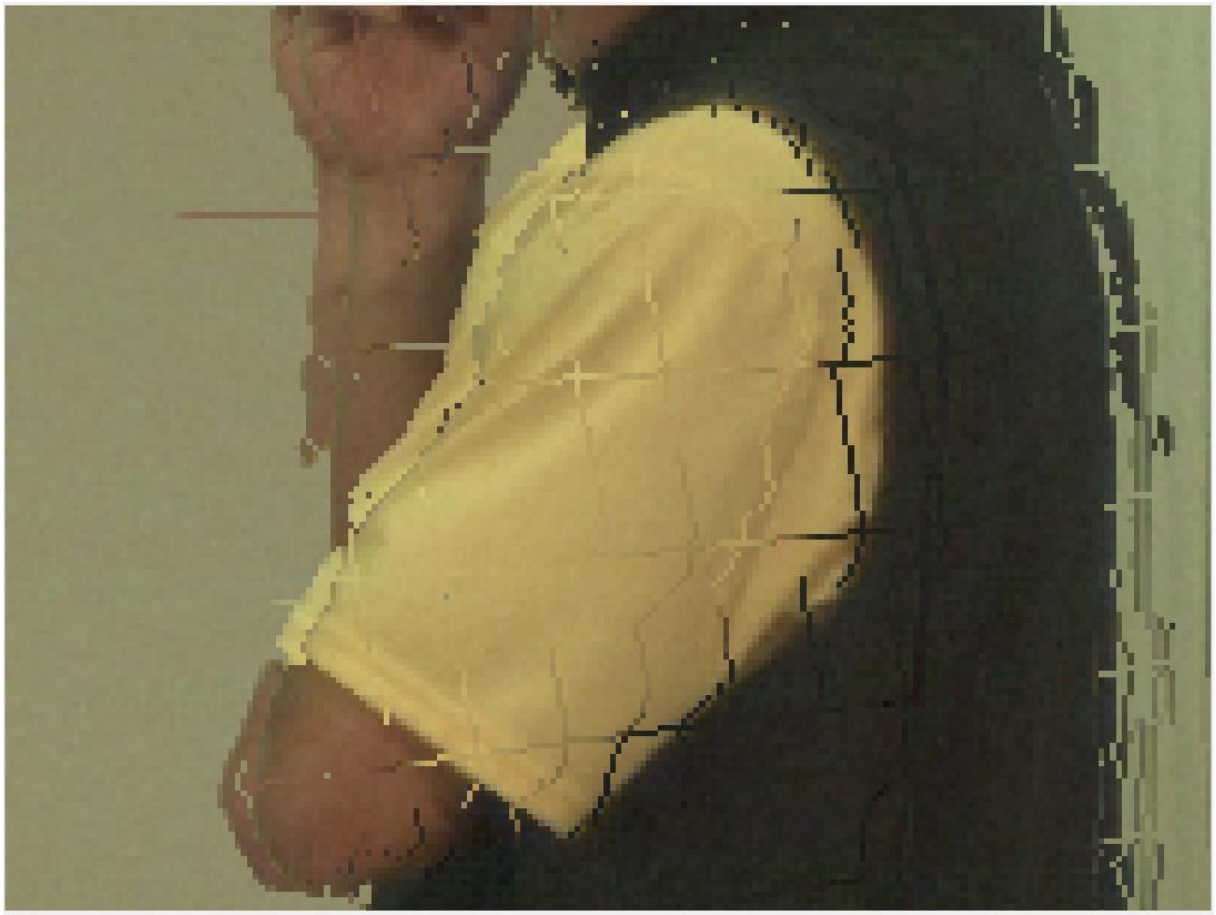
3.5. Filtracja

3.5.1. Idea

Po wypełnieniu dziur w widoku dalej część punktów w obrazie dalej może być reprezentowana niepoprawnie. Przyczyny mogą być dwie:

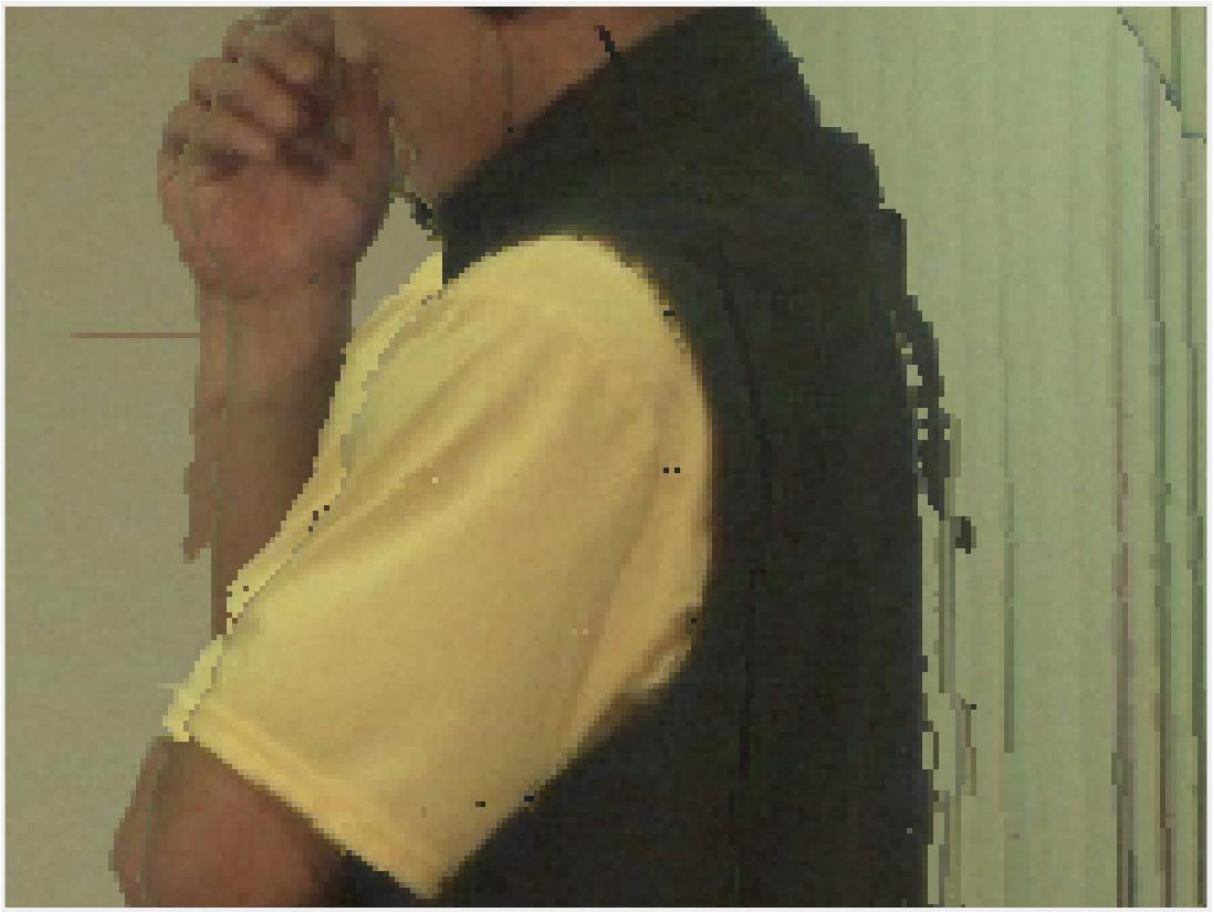
- niektóre punkty zostały niepoprawnie zrzutowane,

- przy wypełnianiu dziur wpisane zostały wartości z obszarów znacząco różnych pod względem koloru.



Rys. 3.6 Błędnie zrzutowane kolory

W celu redukcji tego efektu zastosowana zostaje filtracja [2]. Wyszukiwane są punkty o głębi widocznie różniącej się od sąsiadów. Po zlokalizowaniu takiego punkty na jego miejsce przepisywany jest jego sąsiad. Wykonywane są 2 przejścia po obrazie – poziome (od lewej do prawej) i pionowe (z góry do dołu).



Rys. 3.7 Efekt filtracji

Rys. 3.8 Od lewej: widok wirtualny przed i po zastosowaniu filtracji

3.5.2. Implementacja

```
for h = 1:height
    for w = 1:width
        if(w == 1)
            diffW = (depth(h,w+1)-depth(h,w));
            if(diffW > 2560)
                inpaintedDepth(h,w) = depth(h,w+1);
                inpaintedImage(h,w,1) = image(h,w+1,1);
                inpaintedImage(h,w,2) = image(h,w+1,2);
                inpaintedImage(h,w,3) = image(h,w+1,3);
            end
        elseif(w == width)
            diffW = (depth(h,w-1)-depth(h,w));
            if(diffW > 2560)
                inpaintedDepth(h,w) = depth(h,w-1);
                inpaintedImage(h,w,1) = image(h,w-1,1);
                inpaintedImage(h,w,2) = image(h,w-1,2);
                inpaintedImage(h,w,3) = image(h,w-1,3);
            end
        else
            diffWR = (depth(h,w+1)-depth(h,w));
            diffWL = (depth(h,w-1)-depth(h,w));
            if((diffWR > 2560) && (diffWL > 2560))
                inpaintedDepth(h,w) = depth(h,w+1);
                inpaintedImage(h,w,1) = image(h,w-1,1);
                inpaintedImage(h,w,2) = image(h,w-1,2);
                inpaintedImage(h,w,3) = image(h,w-1,3);
            end
        end
    end
end
```

Kod 3.8 Filtracja cz. 1

W pierwszej pętli dokonywana jest filtracja pozioma. Jeśli głębia danego punktu różni się, od któregoś z sąsiadów (lewego *diffWL* lub prawego *diffWR*) to jego miejsce kopiowany jest jego lewy sąsiad. W przypadku punktów na krawędziach (lewej lub prawej) sprawdzany jest tylko jeden sąsiad.

```
for w = 1:width
    for h = 1:height
        if(h == 1)
            diffH = ((depth(h+1,w)-depth(h,w)));
            if(diffH > 2560)
                inpaintedDepth(h,w) = depth(h+1,w);
                inpaintedImage(h,w,1) = image(h+1,w,1);
                inpaintedImage(h,w,2) = image(h+1,w,2);
                inpaintedImage(h,w,3) = image(h+1,w,3);

            end
        elseif(h == height)
            diffH = ((depth(h-1,w)-depth(h,w)));
            if(diffH > 2560)
                inpaintedDepth(h,w) = depth(h-1,w);
                inpaintedImage(h,w,1) = image(h-1,w,1);
                inpaintedImage(h,w,2) = image(h-1,w,2);
                inpaintedImage(h,w,3) = image(h-1,w,3);
            end
        else
            diffHU = depth(h+1,w)-depth(h,w);
            diffHD = depth(h-1,w)-depth(h,w);
            if((diffHU > 2560) && (diffHD > 2560))
                inpaintedDepth(h,w) = depth(h+1,w);
                inpaintedImage(h,w,1) = image(h-1,w,1);
                inpaintedImage(h,w,2) = image(h-1,w,2);
                inpaintedImage(h,w,3) = image(h-1,w,3);
            end
        end
    end
end
```

Kod 3.9 Filtracja cz.2

Filtracja w kierunku pionowym – działa analogicznie do filtracji poziomej.

4. Synteza z wykorzystaniem korekcji barwnej

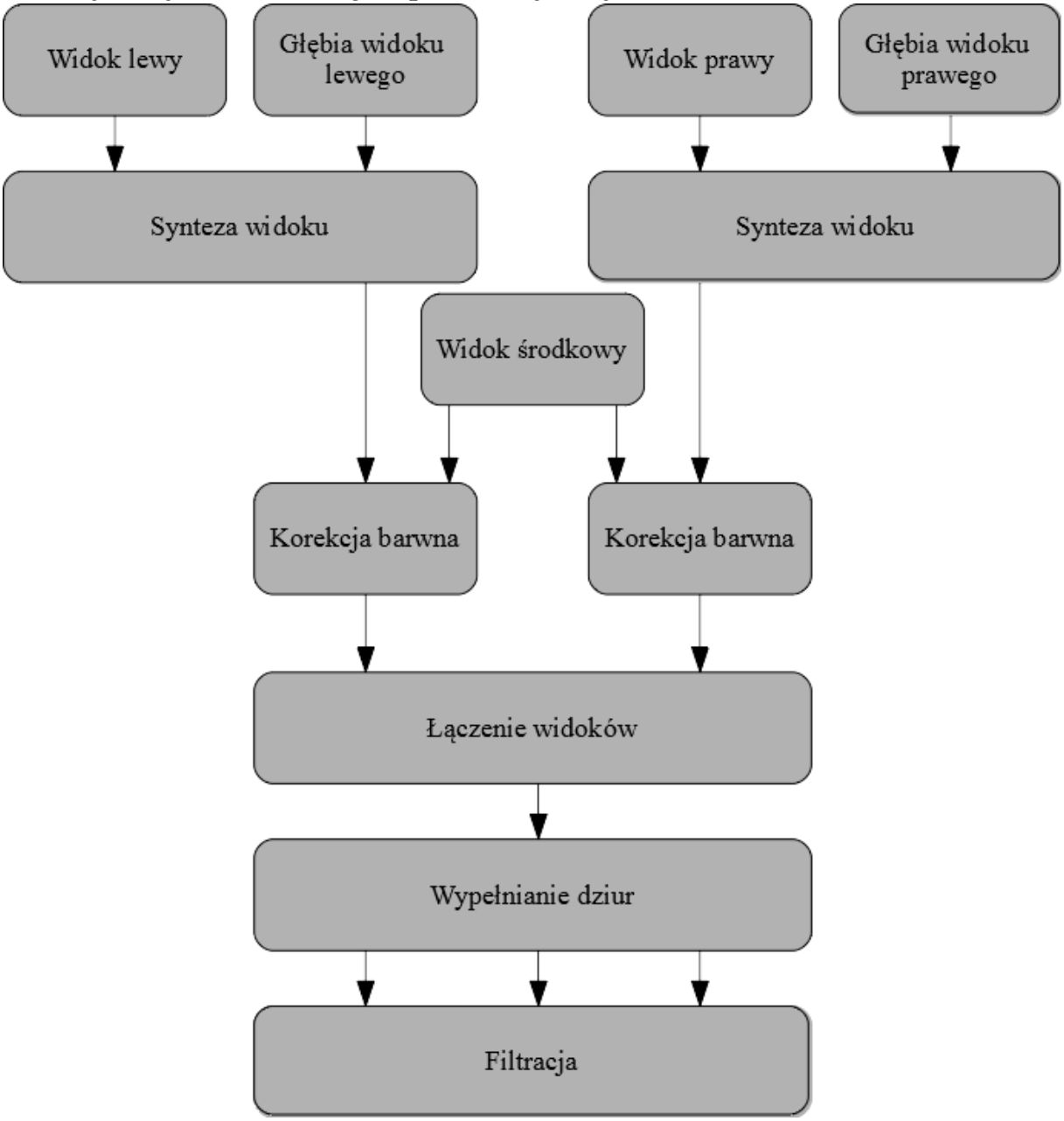
4.1. Niespójności barwne w widokach wirtualnych

Jak wspomniano w rozdziale 1.2 charakterystyki barwne widoków różnią się. Te różnice najbardziej uwidaczniają się po zastosowaniu syntezy.

Rys. 4.1 Przykład niespójności barwnej w widoku wirtualnym

Na powyższym rysunku widzimy niespójności na ścianie – część po lewej jest jaśniejsza od tej po prawej. Wynika to z wcześniej wspomnianych różnic pomiędzy charakterystykami barwnymi w widokach. Ten efekt może prowadzić do dyskomfortu u widza, dlatego uznany jest za niepożądany i należy go zredukować.

4.2. Wykorzystanie korekcji w procesie syntezy



Rys. 4.2 Synteza widoków wirtualnych z wykorzystaniem korekcji barwnej

Syntezę widoków wirtualnych wzbogacono o algorytm korekcji barwnej. Widoki powstałe w wyniku syntezy widoków lewego i prawego przed łączeniem poddawane są korekcji względem widoku środkowego służącego za obraz odniesienia. Celem takiego działania jest wyrównanie charakterystyk barwnych widoków po syntezie co powinno zredukować niespójności w widoku otrzymywanym na końcu syntezy.

4.3. Testy

W syntezie wykorzystano 4 sekwencje:

- *Ballet* (100 ramek w sekwencji, rozdzielczość 1024 punkty na 768 linii),

- *BBB Flowers Arc* (120 ramek w sekwencji, rozdzielczość 1280 punkty na 768 linii),

- *PoznanFencing* (250 ramek w sekwencji, rozdzielczość 1920 punkty na 1080 linii),

- *SoccerArc* (250 ramek w sekwencji, rozdzielczość 1920 punkty na 1080 linii).

Rys. 4.3 Obraz z sekwencji *Ballet*

Rys. 4.4 Obraz z sekwencji *BBB Flower Arc*

Rys. 4.5 Obraz z sekwencji *Soccer Arc*

Rys. 4.6 Obraz z sekwencji *PoznanFencing*

4.3.1. Testy obiektywne

W ramach testów obiektywnych obliczono IV-PSNR (*Immersive Video Peak Signal to Noise Ratio*)[3]. IV-PSNR to metryka oparta o PSNR (*Peak Signal to Noise Ratio*) dostosowana do zastosowań w wizji wszechogarniającej. W porównaniu do PSNR IV-PSNR eliminuje wpływ przesunięcia krawędzi obiektów (spowodowanych niedokładnym rzutowaniem w syntezie).

4.3.2. Testy subiektywne

Ze względu na sytuację epidemiologiczną w Polsce (pandemia COVID-19) trwającą w czasie powstawania pracy testy subiektywne odbyły się w formie zdalnej w celu minimalizacji niebezpieczeństw dla przeprowadzających testy i osób ankietowanych.

W ramach testów subiektywnych grupie ochotników udostępniono ankiety z zestawem sekwencji porównawczych (metoda *Pair Comparison* rekomendowana przez ITU-T w ITU-T P.910). Składał się z 4 podzestawów po 6 porównań.

Sekwencja porównawcza – zestawienie dwóch sekwencji wizyjnych, jednej bez korekcji, a drugiej skorygowanej wybraną metodą. Wszystkie użyte sekwencje składały się dokładnie z 250 obrazów odtwarzanych z prędkością ok. 25 obrazów/sekunda. Sekwencje *Ballet* oraz *BBB Flowers Arc* należało przedłużyć do wymaganej długości, a dokonano tego poprzez dodatkowe odtworzenie ich „od tyłu", a potem dodanie brakującej liczby obrazów we właściwej kolejności (przykład: sekwencja 3-ramkowa przedłużona do 7 ramek będzie się odtwarzać w następującej kolejności – 1 2 3 2 1 2 3).

Podzestaw składał się z 6 sekwencji porównawczych:

- Brak korekcji | Algorytm liniowy,

- Brak korekcji | Algorytm przeniesienia koloru,

- Brak korekcji | Algorytm wyrównania histogramów,

- Algorytm liniowy | Brak korekcji,

- Algorytm przeniesienia koloru | Brak korekcji,

- Algorytm wyrównania histogramów | Brak korekcji.

Uczestnik otrzymywał link do odtworzenia sekwencji porównawczej, a następnie był proszony o określenie czy któraś z sekwencji wygląda lepiej, a jeśli tak to w jakim stopniu (trochę lepiej, lepiej lub dużo lepiej). Uczestnik nie wiedział, na której sekwencji zastosowano korekcję oraz jakiego typu. Jeśli pojawiały się niespójności w odpowiedziach, np. w jednym pytaniu zaznaczono, że korekcja poprawia jakość, a w pytaniu z zamienioną kolejnością sekwencji zaznaczono, że pogarsza, to dana ankieta mogła zostać odrzucona. W celu uzyskania wiarygodniejszych wyników każdy uczestnik otrzymywał pytania w innej losowej kolejności.

Poniżej znajdują się kadry z sekwencji porównawczych (za każdym razem sekwencja bez korekcji po lewej) wraz z omówionymi różnicami.

Rys. 4.7 *Ballet* – przeniesienie koloru

Rys. 4.8 *Ballet* – algorytm liniowy

Rys. 4.9 *Ballet* – algorytm wyrównania histogramów

W przypadku sekwencji *Ballet* różnice są widoczne na ścianie za tancerką. Mniej więcej w 1/3 ramki (okolice głowy tancerki) widać granicę łączenia dwóch widoków – lewa strona ściany jest jaśniejsza, a prawa ciemniejsza. Po zastosowaniu algorytmów korekcji te różnice zostały zniwelowane.

Rys. 4.10 *BBB Flowers Arc* – przeniesienie koloru

Rys. 4.11 *BBB Flowers Arc* – algorytm liniowy

Rys. 4.12 *BBB Flowers Arc* – algorytm wyrównania histogramów

W sekwencji *BBB Flowers Arc* nie widać dużych niespójności oraz różnic pomiędzy korekcję, a jej brakiem.

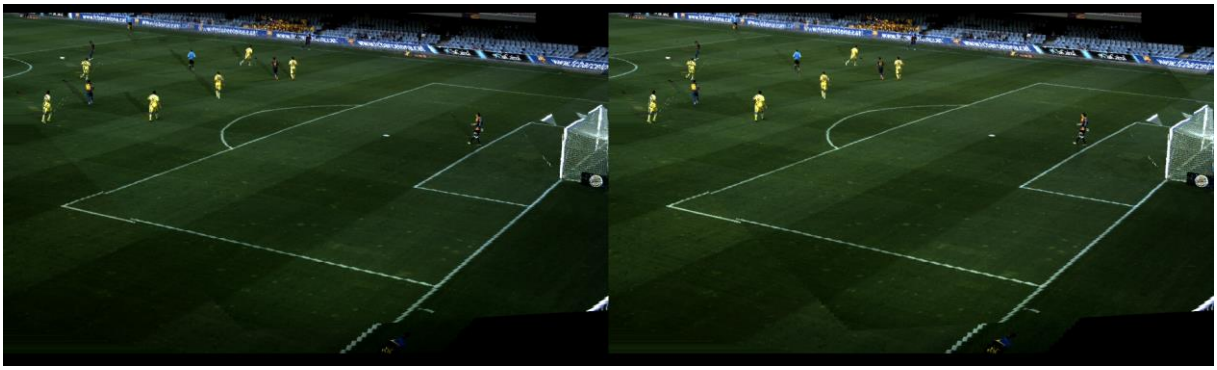
Rys. 4.13 *PoznanFencing* – przeniesienie koloru

Rys. 4.14 *PoznanFencing* – algorytm liniowy

Rys. 4.15 *PoznanFencing* – algorytm wyrównania histogramów

W sekwencji *PoznanFencing* sytuacja jest zbliżona do tej w *Ballet*, czyli różnice są widoczne na ścianach (dodatkowo występują one na ubraniach szermierzy). Korekcja widocznie pojaśnia tło i ubrania w ten sposób usuwając niespójności.

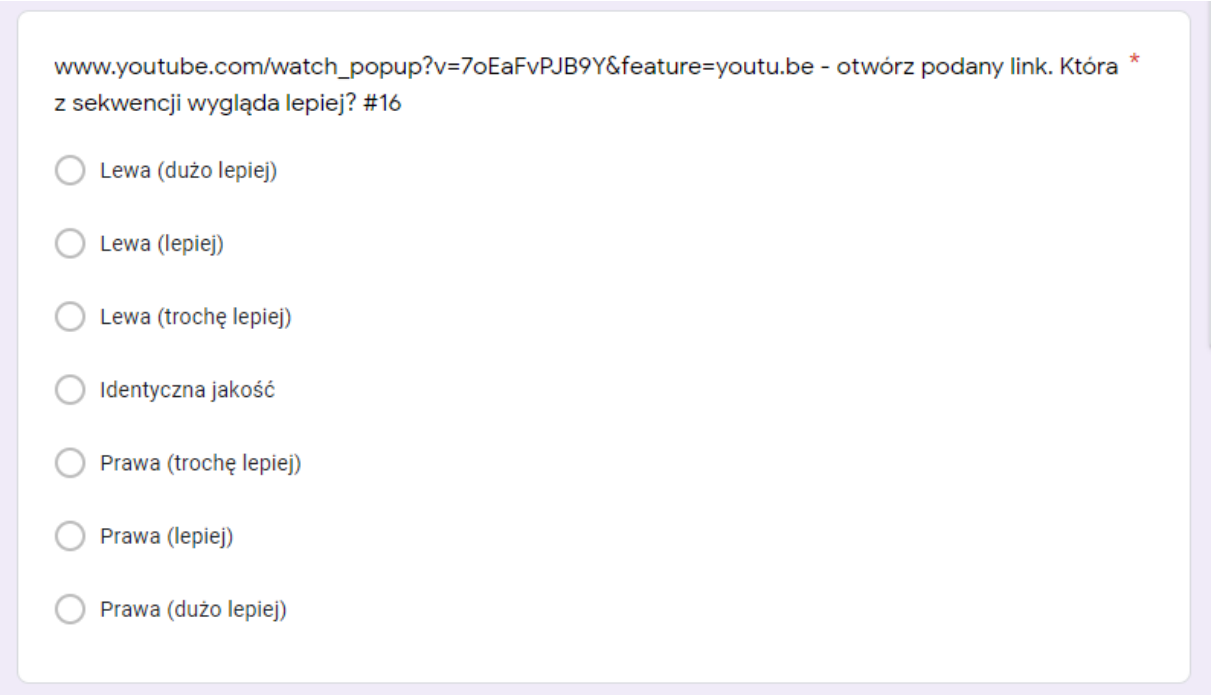
Rys. 4.16 *Soccer Arc* – przeniesienie koloru



Rys. 4.17 *Soccer Arc* – algorytm liniowy

Rys. 4.18 *Soccer Arc* – algorytm wyrównania histogramów

Sekwencja *Soccer Arc* jest stosunkowo spójna barwnie (w porównaniu do *Fencing*), różnice są widoczne w lewym dolnym rogu ramki gdzie widać granicę łączenia widoków (lewa dolna część jest ciemniejsza od reszty ramki). Korekcja pojaśnia sekwencję, dodatkowo przeniesienie koloru w lewym górnym rogu tworzy „siatkę" czarnych punktów (punktów, które były wypełniane w procesie syntezy).



Rys. 4.19 Przykładowe pytanie w ankiecie

4.4. Wyniki testów

4.4.1. Testy obiektywne

		IV-PSNR
Ballet	Brak korekcji	36,24
	Przeniesienie koloru	36,01
	Algorytm liniowy	36,15
	Wyrównanie hist.	35,89
BBB Flowers Arc	Brak korekcji	25,15
	Przeniesienie koloru	25,08
	Algorytm liniowy	25,14
	Wyrównanie hist.	25,12
PoznanFencing	Brak korekcji	26,83
	Przeniesienie koloru	27,03
	Algorytm liniowy	27,26
	Wyrównanie hist.	27,25
Soccer Arc	Brak korekcji	28,28
	Przeniesienie koloru	28,61
	Algorytm liniowy	28,58
	Wyrównanie hist.	28,29

Tab. 4.1 Wyniki testów obiektywnych cz. 1

	IV-PSNR
Brak korekcji	29,12
Przeniesienie koloru	29,18
Algorytm liniowy	29,28
Wyrównanie hist.	29,13

Tab. 4.2 Wyniki testów obiektywnych cz. 2

	IV-PSNR
Ballet	36,07
BBB Flowers Arc	25,12
PoznanFencing	27,09
Soccer Arc	28,44

Tab. 4.3 Wyniki testów obiektywnych cz. 3

Można zauważyć, że korekcja w bardzo niewielki sposób wpływa na zmianę IV-PSNR. Dla sekwencji *Ballet* i *BBB Flowers Arc* IV-PSNR zmalało po zastosowaniu korekcji, z kolei dla pozostałych IV-PSNR wzrosło przy czym w obu przypadkach zmiana nie przekraczała 0,5 dB (maksymalna różnica: 0,43 dB, średnia: 0,21 dB).

4.4.2. Testy subiektywne

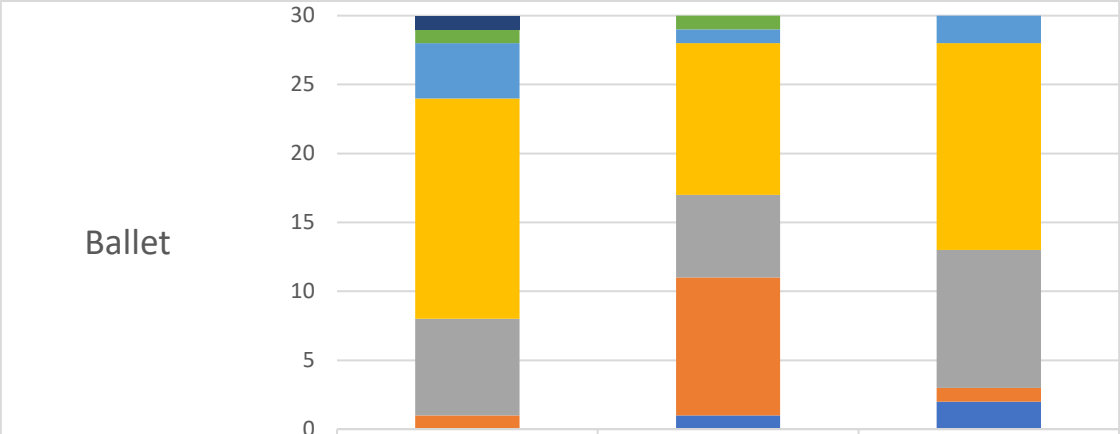
Uzyskano łącznie 45 wypełnionych ankiet z czego 15 odrzucono ze względu na duże niespójności w odpowiedziach (co najmniej 1/3 niespójnych odpowiedzi).

		Korekcja poprawia jakość	Identyczna jakość	Korekcja pogarsza jakość
Ballet	Przesunięcie koloru	8	16	6
	Alg. Liniowy	17	11	2
	Wyrównanie hist.	13	15	2
BBB Flowers Arc	Przesunięcie koloru	4	21	5
	Alg. Liniowy	7	18	5
	Wyrównanie hist.	9	16	5
PoznanFencing	Przesunięcie koloru	27	3	0
	Alg. Liniowy	28	1	1
	Wyrównanie hist.	27	0	3
Soccer Arc	Przesunięcie koloru	11	6	12
	Alg. Liniowy	8	12	10
	Wyrównanie hist.	12	7	11

Tab. 4.4 Wyniki testów subiektywnych cz. 1

		Śr. ważona	Śr. ważona (cała sekwencja)
Ballet	Przesunięcie koloru	0,00	
	Alg. Liniowy	0,87	
	Wyrównanie hist.	0,53	0,47
BBB Flowers Arc	Przesunięcie koloru	-0,10	
	Alg. Liniowy	0,07	
	Wyrównanie hist.	0,13	0,03
PoznanFencing	Przesunięcie koloru	1,53	
	Alg. Liniowy	1,73	
	Wyrównanie hist.	1,37	1,54
Soccer Arc	Przesunięcie koloru	-0,23	
	Alg. Liniowy	-0,10	
	Wyrównanie hist.	0,10	-0,08

Tab. 4.5 Wyniki testów subiektywnych cz.2



	Przesunięcie koloru	Alg. Liniowy	Wyrównanie hist.
Korekcja znacznie pogarsza	1	0	0
Korekcja pogarsza	1	1	0
Korekcja trochę pogarsza	4	1	2
Identyczna jakość	16	11	15
Korekcja trochę poprawia	7	6	10
Korekcja poprawia	1	10	1
Korekcja znacznie poprawia	0	1	2

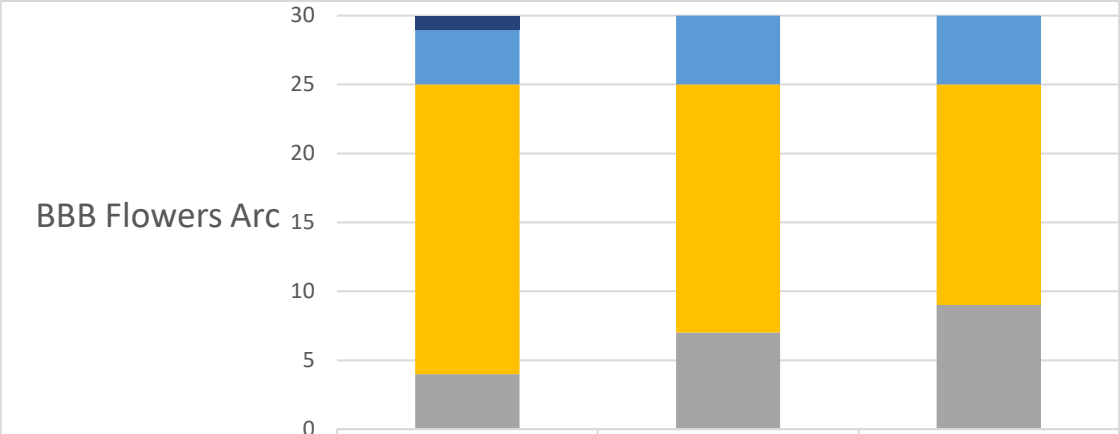
Rys. 4.20 Szczegółowe wyniki dla sekwencji *Ballet*

Dla sekwencji *Ballet:*

- przesunięcie koloru w większości przypadków nie daje widocznej różnicy w jakości, jeśli występują to różnice są niewielkie,

- w przypadku algorytmu liniowego różnice są najbardziej obecne i w większości przypadków obecna jest widoczna poprawa jakości,

- algorytm wyrównania histogramów daje największą ilość drobnych zmian na korzyść przy czym i tak dominuje brak widocznych zmian.



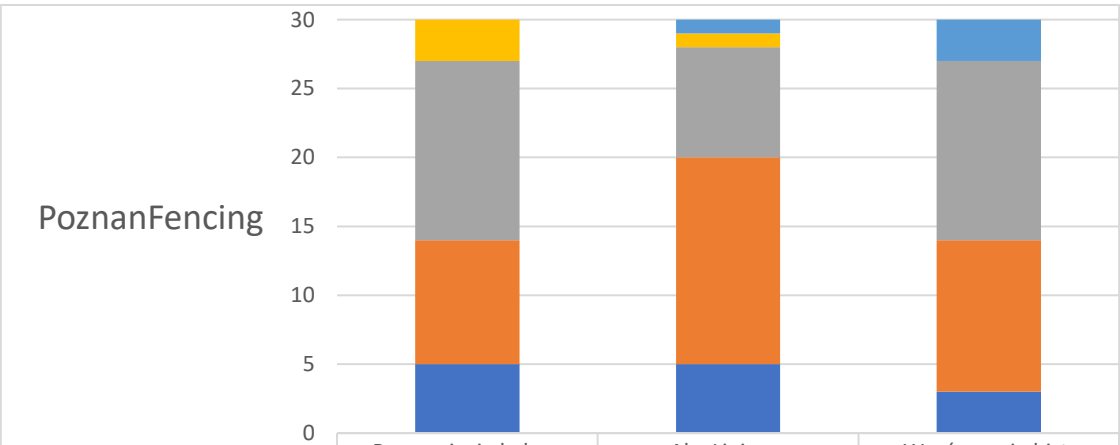
	Przesunięcie koloru	Alg. Liniowy	Wyrównanie hist.
Korekcja znacznie pogarsza	1	0	0
Korekcja pogarsza	0	0	0
Korekcja trochę pogarsza	4	5	5
Identyczna jakość	21	18	16
Korekcja trochę poprawia	4	7	9
Korekcja poprawia	0	0	0
Korekcja znacznie poprawia	0	0	0

Rys. 4.21 Szczegółowe wyniki dla sekwencji *BBB Flowers Arc*

Dla sekwencji *BBB Flowers Arc*:

- korekcja nie poprawia ani nie pogarsza jakości w widoczny sposób,

- znacząco dominuje brak zmiany jakości.



	Przesunięcie koloru	Alg. Liniowy	Wyrównanie hist.
Korekcja znacznie pogarsza	0	0	0
Korekcja pogarsza	0	0	0
Korekcja trochę pogarsza	0	1	3
Identyczna jakość	3	1	0
Korekcja trochę poprawia	13	8	13
Korekcja poprawia	9	15	11
Korekcja znacznie poprawia	5	5	3

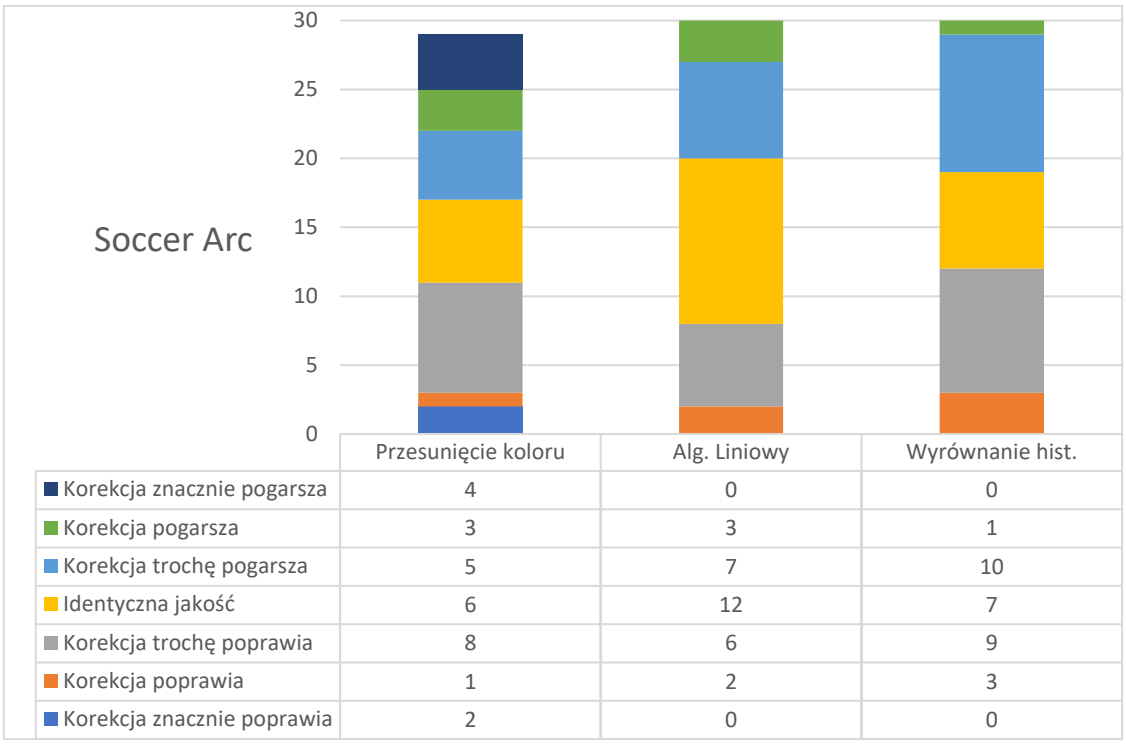
Rys. 4.22 Szczegółowe wyniki dla sekwencji *PoznanFencing*

Dla sekwencji *PoznanFencing*:

- korekcja poprawia jakość we wszystkich przypadkach (mało liczne opinie o pogorszeniu jakości)

- najlepsze wyniki otrzymano dla algorytmu liniowego gdzie dominuje widoczna poprawa jakości z opiniami o znacznej poprawie jakości,

- nieco gorsze wyniki dla algorytmu wyrównania histogramów choć wciąż obecna jest poprawa jakości,



	Przesunięcie koloru	Alg. Liniowy	Wyrównanie hist.
Korekcja znacznie pogarsza	4	0	0
Korekcja pogarsza	3	3	1
Korekcja trochę pogarsza	5	7	10
Identyczna jakość	6	12	7
Korekcja trochę poprawia	8	6	9
Korekcja poprawia	1	2	3
Korekcja znacznie poprawia	2	0	0

Rys. 4.23 Szczegółowe wyniki dla sekwencji *Soccer*

Dla sekwencji *Soccer Arc*:

- przesunięcie koloru daje najbardziej niejednoznaczne wyniki, w większości przypadków różnice są rozróżniane, ale nie można jednoznacznie określić ich stopnia oraz czy zmiana jest pozytywna bądź negatywna,

- algorytm wyrównania histogramów daje podobne wyniki do algorytmu przesunięcia koloru przy czym różnice są mniej rozrzucone,

- algorytm liniowy daje najmniej skrajne wyniki w większości skłaniające się ku braku zmiany jakości.

5. Wnioski

5.1. Testy obiektywne

Korekcja nie wpływa znacząco na IV-PSNR, gdyż celem nie jest odtworzenie dokładnej charakterystyki widoku wirtualnego, a wyrównanie charakterystyk dwóch widoków wejściowych. Korekcja nie wpływa znacząco na charakter i liczbę artefaktów w widokach. Wynika to z tego, że korygowane widoki, pomimo niespójności barwnych, są do siebie podobne, a korekcja nie ma za zadanie usunąć artefaktów, a jedynie zwiększyć spójność barwną. Przekłada się to na niewielkie zmiany IV-PSNR.

5.2. Testy subiektywne

W sekwencji *Ballet* dla algorytmu przeniesienia koloru widzowie w większości wskazali, że korekcja nie zmienia jakości, a jeśli tak to w niewielkim stopniu. Podobne wyniki uzyskano dla algorytmu wyrównania histogramów z wskazaniem, że jeśli było widać różnicę to poprawiające jakość. Najlepsze wyniki uzyskano dla algorytmu liniowego – tutaj większość opinii wskazywała na poprawę jakości. Wynika to najprawdopodobniej z tego, że elementy/postacie pierwszego planu sceny są spójne barwnie. Różnice są widoczne w tle (w tym wypadku ściana) jednak tutaj 1/3 ramki jest przerzutowana z jednego widoku, a reszta z drugiego (widoczna jest granica łączenia widoków). Widać, że jeden widok zdominował większość ramki, a pozostała jej część jest zrzutowana z drugiego widoku. Poza granicą łączenia widoków te 2 części są spójne (brak „płatów" z jednego widoku na części zdominowanej przez drugi widok). Dodatkowo różnice pomiędzy dwoma widokami nie są aż tak duże. Algorytm przeniesienia koloru operuje na średniej i odchyleniu standardowym, a w przypadku stosunkowo podobnych barwnie widoków korekcja nie zmienia aż tak dużo. Algorytmy liniowy i wyrównania histogramów operują na histogramach dlatego drastyczniej zmieniają widoki (co wpływa pozytywnie na spójność).

W sekwencji *BBB Flowers* wszystkie sekwencji źródłowe zostały wygenerowane komputerowo. Widzowie wskazali, że korekcja nie zmienia jakości. Wynika to najprawdopodobniej z tego, że nie występują różnice w parametrach kamer (skoro sekwencja jest wygenerowana komputerowo to parametry kamer nie będą się różnić) oraz scena jest równomiernie oświetlona. Możliwe, że gdyby scena była bardziej dynamicznie oświetlona albo zastosowana została technika symulacji zachowania światła jak śledzenie wiązki (*Raytracing*) bądź śledzenia ścieżki (*Pathtracing*) pojawiłyby się niespójności do skorygowania.

W sekwencji *PoznanFencing* widzowie wskazali, że w każdym przypadku korekcja poprawia jakość. Tutaj nie ma sytuacji analogicznej w *Ballet*, czyli że część ramki została zdominowana przez jeden widok, a reszta przez drugi, obie sekwencja nakładają się na siebie i punkty głębi mają zbliżone wartości, dlatego występują „płaty" z jednego widoku otoczone obszarem z drugiego widoku. W sytuacji, gdy oba widoki znacznie się różnią pod względem charakterystyk barwnych (w jednym ściana jest jaśniejsza, a drugim ciemniejsza).

W sekwencji *Soccer Arc* mamy sytuację analogiczną do tej w *Ballet*, czyli większość ramki zdominowana jest przez jeden widok, a reszta przez drugi (lewy dolny róg ramki), dlatego korekcja nie daje widocznej poprawy jakości. Algorytm przeniesienia koloru

spowodował, że uwidoczniły się linie, które powinny być usunięte w wypełnianiu dziur co spowodowało widoczne pogorszenie jakości. Algorytmy liniowy i wyrównania histogramów dały niejednoznaczne wyniki. Różnice pomiędzy wspomnianymi zdominowanymi obszarami nie są duże, a 2 wspomniane algorytmy usuwają owe różnice jednocześnie „pojaśniając" widok uzyskany w wyniku syntezy. Najprawdopodobniej to spowodowało takie wyniki ankiet – widzowie nie zwracali bardzo uwagi na różnice pomiędzy dwoma zdominowanymi obszarami i nie potrafili potem jednoznacznie określić czy sekwencja „jaśniejsza" wygląda lepiej.

Można zauważyć, że im jaśniejsze jest tło sceny tym widoczniejsze stają się niespójności barwne. Jeśli nie występuje sytuacja, że część ramki jest zdominowana przez jeden widok, a druga część przez drugi to niespójności powinny być lepiej widoczne, a korekcja dawać lepszą poprawę jakości. Możliwe, że gdyby doszło do sytuacji, w której ramka jest podzielona mniej więcej 50% z jednego widoku, 50% z drugiego to różnice byłyby bardziej widoczne.

Algorytm liniowy dał najlepsze wyniki, algorytm przeniesienia koloru najgorsze. Trochę gorsze wyniki od algorytmu liniowego, ale wciąż dużo lepsze od przeniesienia koloru uzyskano dla wyrównania histogramów. Słabość przeniesienia koloru najprawdopodobniej wynika z tego, że bazuje ono na średniej arytmetycznej oraz odchyleniu standardowym z wartości charakterystyk barwnych. Algorytmy liniowy i wyrównania histogramów operują na analizie histogramów składowych barwnych. Prawdopodobnie lepsza efektywność algorytmu liniowego wynika z tego, że modyfikuje on punkty obrazu (czyt. mnoży przez współczynnik korekcji), a wyrównanie histogramów przekopiowuje i przenosi wartości punktów według funkcji M (jeśli różnice pomiędzy obrazami będą duże to nie będą występowało odpowiednio podobne wartości do powielenia).

Bibliografia

[1] „Korekcja barwna sekwencji wielowidokowych”, A. Kuehn, praca magisterska, 2015

[2] „Synteza widoków wirtualnych w rzadkich systemach wielokamerowych dla zastosowań w swobodnej nawigacji”, A. Dziembowski, praca doktorska, 2018

[3] „Software manual of IV-PSNR for Immersive Video”, A. Dziembowski, ISO/EIC JTC1/SC29 WG04/MPEG N0013, Online, 2020

[4] „Obraz cyfrowy”, M. Domański, Wydawnictwo Komunikacji i Łączności, 2010

[5] „Performance Evaluation of Color Correction Approaches for Automatic Multi-view Image and Video Stitching”, W.Xu, J. Mulligan, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), San Francisco, USA, 2010

[6] „Akwizycja sekwencji wielowidokowych”, A. Kuehn, A. Dziembowski, praca inżynierska, 2013

[7] „Telewizja swobodnego punktu widzenia – nowa usługa czy futurystyczna wizja”, M.Domański, A. Dziembowski, D. Mieloch, A. Łuczak, O. Stankiewicz, K. Wegner, Przegląd Telekomunikacyjny, nr 8-9, 2014

[8] „Color Transfer between Images” E. Reinhard, M. Ashikhmin, B. Gooch, P. Shirley, IEEE Computer Graphics and Applications, vol. 21, 2001

[9] „Histogram-Based Prefiltering for Luminance and Chrominance Compensation of Multiview Video Coding” U. Fecker, M. Barkowsky, A. Kaup, IEEE Transactions on Circuits and Systems for Video Technology, nr 9, 200

[10] „Demonstration of a simple free viewpoint television system”, M. Domański, A. Dziembowski, T. Grajek, A. Grzelka, K. Klimaszewski, D. Mieloch, R. Ratajczak, O. Stankiewicz, J. Siast, J. Stankowski, K. Wegner, IEEE International Conference on Image Processing, Pekin, Chiny, 2017

[11] „Real-time virtual navigation provision by simple means”, M. Domański, A. Dziembowski, T. Grajek, A. Grzelka, D. Mieloch, O. Stankiewicz, J. Stankowski, K. Wegner, International Conference on Signals and Electronic Systems, Kraków, Polska, 2018