Politechnika Warszawska

Studia magisterskie

Cyfrowe Przetwarzanie Obrazu (2D)

DOKUMENTACJA

Program do malowania światłem na materiale video

inż. Jarosław Affek
nr indeksu: 269259

Warszawa 2020

1. Wstęp

Celem projektu było stworzenie programu, który umożliwiłby „malowanie” światłem, czyli uzyskanie efektu znanego z fotografii pod hasłem „light painting”, z tą różnicą, że proces malowania uwieczniony byłby w postaci materiału video, a nie pojedynczej fotografii z długim czasem naświetlania. Dzięki czemu widoczny byłby proces powstawania smug świetlnych. Do pisania programu świadomie użyto języka programowania *Python*, pomimo że nie jest on najwydajniejszy w przypadku ręcznego przetwarzania obrazów (bez użycia sieci neuronowych) ze względu na konieczność jego nauki w perspektywie pisania pracy magisterskiej. Do przetwarzania obrazu wykorzystana została biblioteka *OpenCV*, a do niektórych operacji macierzowych biblioteka *Numpy.* W trakcie tworzenia programu niektóre pomysły i algorytmy okazywały się lepsze lub gorsze, jednak nie zrezygnowano z żadnego z nich, aby umożliwić uzyskanie jak najbardziej zróżnicowanych efektów z jednego materiału źródłowego. Pozostawiono wszystkie opcje wyboru parametrów użytkownikowi.

2. Algorytmy detekcji światła

Aby uzyskać pożądany efekt, należało znaleźć sposób na wyodrębnienie i kumulowanie lub kopiowanie światła (wartości jasnych pikseli) w kolejnych klatkach materiału video. Rozważono i zaimplementowano następujące metody:

- ***Simple sum*** **- prosta kumulacja wartości wszystkich pikseli w czasie**

Najprostsza i dająca najgorsze rezultaty metoda polega na sumowaniu wartości każdego piksela (każdego z kanałów RGB niezależnie) wszystkich kolejnych klatek filmu. Metoda ta ma szansę dać dobre rezultaty tylko w przypadku idealnie czarnego tła, wtedy sumowane będą tylko wartości niezerowych pikseli. Jednak nieprzetworzone video prosto z kamery nigdy nie będzie miało czarnego tła, więc w kolejnych klatkach szum i artefakty tła będą się kumulować i psuć obraz.

- ***Thresholding*** **- wykorzystanie progu jasności**

W tej metodzie następuje sumowanie tylko pikseli o intensywności przekraczającej pewien próg. Dzięki temu tworzona jest maska na każdej kolejnej klatce filmu i do sumy poprzednich klatek dodawany tylko ten fragment nowej klatki, w którym piksele osiągają co najmniej progową jasność.

- ***Background substraction*** **- wykorzystywanie odejmowania tła**

Jest to najbardziej zaawansowana metoda oparta na funkcji *BackgroundSubstractor* w *OpenCV*. Metoda ta wymaga, aby początek materiału filmowego stanowiło jedynie nieruchome tło (min. jedna klatka), zanim nastąpi malowanie światłem. Funkcja działa w taki sposób, że na każdej kolejnej klatce „uczy” się wyglądu tła (poprzez operacje segmentacji z użyciem metod Gaussa). Następnie w momencie rozpoczęcia malowania, w programie ustawiany jest współczynnik uczenia funkcji na 0, aby klatki ze światłem już nie wpływały na proces uczenia tła. Funkcja analizując każdą kolejną klatkę jest w stanie wyznaczyć maskę w miejscach, gdzie kolejne klatki różnią się od tła. Kolejno następuje sumowanie kolejnych

klatek uwzględniając maskę jak w metodzie z progiem. Dopasowanie wyznaczanych masek w tej metodzie jest silnie zależne od podanych parametrów do funkcji *BackgrounSubstractor*, użytkownik ma możliwość ingerencji w parametry uczenia. Po odpowiednim ustawieniu metoda ta daje zdecydowanie najlepsze rezultaty.

Przy wszystkich powyższych metodach do standardowego działania wymagane jest nieruchoma kamera podczas nagrywania. Jednak celowy ruch kamery może wprowadzić pewne trudne do przewidzenia artystyczne efekty.

3. Algorytmy sumowania światła

Standardowe sumowanie wartości pikseli omówione powyżej nie wymaga komentarza, jednak podczas testowania programu stwierdzono, że dodatkowym interesującym efektem może być sumowanie światła, ale tylko z pewnej ograniczonej liczby klatek. Aby to uzyskać dodano parametr *Trail length*, który określa z ilu klatek mają się składać smugi światła. Da to efekt zanikania śladów światła po określonym czasie. Sprawdzono i zaimplementowano następujące metody sumowania:

- ***Copy***

Proste kopiowanie – na sumę z poprzednich klatek nakładana jest nowa (z uwzględnieniem maski), jeśli maska jest poprawnie wyliczona (dotyczy algorytmów *Background substraction* i *Thresholding*) to efekt jest dobry, jednak w przypadku niedopasowania maski powstają artefakty i „przerwy" w smudze światła. Piksele, które obejmuje maska z nowej klatki zastępują piksele z poprzedniej. Proste kopiowanie nie ma sensu w przypadku algorytmu *Simple sum* gdyż nie uwzględnia on masek i po prostu nowa klatka w całości jest nakładana na poprzednią, co sprawia, że na wyjściu otrzymujemy taki sam materiał video jak na wejściu. Ze względu na pewne ograniczenia biblioteki *OpenCV* w *Pythonie* kopiowanie z uwzględnieniem maski odbywa się „ręcznie", a nie z wykorzystaniem predefiniowanej funkcji, dlatego czas obliczeń jest dość długi.

- ***Accumulate***

Ten tryb wykorzystuje funkcję *accumulate* w *OpenCV*, która działa w taki sposób, że wkleja (uwzględniając maskę) elementy z nowej klatki na poprzednią, sumując (a nie zastępując jak powyżej) wartości pikseli. Dzięki temu otrzymywany efekt jest bardziej naturalny i zbliżony do działania matrycy aparatu (dwukrotne naświetlenie tych samych pikseli powoduje ich rozjaśnienie, a nie zamianę wartości).

- ***Fading copy***

Działa analogicznie jak tryb *Copy*, ale kolejne klatki mają swoje wagi. Tryb ten dostępny jest w przypadku ograniczonej długości śladu światła (skończona wartość parametru *Trail length*), przykładowo dla śladu o długości 25 klatek to waga ostatniej wczytanej klatki wynosi 1 (zapisywana z pełną jasnością), a 25 klatka licząc wstecz ma wagę 1/25. Dzięki temu otrzymywany jest efekt zanikania. Wymaga to przetrzymywania w typie *lista* tylu obrazów (wraz z maskami) ile wynosi wartość parametru *Trail length*. Przez to każda ostatecznie zapisywana klatka jest złożeniem wszystkim obrazów z listy. Sprawia to, że obrazy są generowane dość długo.

- ***Weighted accumulate (OpenCV function)***

Tryb wykorzystuje gotową funkcję *accumulateWeighted* z *OpenCV* jej efekty są bardzo zbliżone do trybu *Fading copy*, ale działa nieco szybciej. Również w tym przypadku do funkcji przekazywane są wyliczone wagi poszczególnych klatek.

- ***Fading accumulation***

Połączenie funkcjonalności trybu *Accumulate* oraz *Fading copy*. Efektem jest zanikający w czasie ślad oraz naturalny efekt sumowania się świateł.

- ***Average***

Tryb ten wymaga określenia skończonej wartości parametru *Trail length.* Wartość pikseli w każdej klatce jest dzielona przez liczbę klatek w danym śladzie światła. W przypadku długiego śladu np. 100 klatek jasność wynosiłaby jedynie 1/100 oryginalnego, więc określono regulowany przez użytkownika parametr wzmocnienia Average g*ain*, który zwiększa równomiernie „udział" każdej klatki w finalnym obrazie. Wartości tego parametru wahają się od 0 do 1, gdzie 0 oznacza wartość 1/100 (dla przedstawionego przykładu), a 1 oznacza jasność wejściową, wtedy efekt działania jest analogiczny jak w trybie *Copy.*

4. Implementacja

- **klasa *Echo* oraz dziedziczące: *standard_echo* i *inverted_echo***

Niezależnie od wybranego trybu w programie zawsze tworzona jest instancja klasy *standard_echo* lub *inverted_echo* dziedziczącej po klasie *Echo*, która podczas działania programu przechowuje **listę klatek filmu potrzebnych do generowania śladu o określonej długości**, model tła oraz kilka parametrów takich jak długość śladu, wybrany tryb sumowania oraz zmienna typu bool określająca czy ślad ma być trwały czy o skończonej długości. W klasie *Echo* znajdują się też niezbędne funkcje do obsługi tych parametrów:

- *update*, funkcja która aktualizuje stan listy z klatkami (po wczytaniu nowej klatki umieszcza ją na końcu listy a usuwa najstarszą).
- *iterate*, funkcja dziedziczona, która służy do przejścia po całej długości listy i stworzenia na jej podstawie klatki, która ma zostać zapisana – odbywa się to w różny sposób, zależnie od wybranego trybu sumowania. Jest dziedziczona, ponieważ zależnie od wybranego trybu ślad może zanikać „od przodu" lub „od tyłu" (*reversed*) – iteracja od początku lub od końca listy
- *paint_function*, funkcja wywołująca funkcję *iterate*, oraz funkcje normalizujące wartości pikseli i konwertujące obrazek na odpowiedni typ, czyli *uint8*

```
class Echo():
  def __init__(self, trail_length, frame_width, frame_height,show_last_frame,painting_style):
    self.background = np.zeros((frame_height,frame_width,3),dtype=np.uint8)
    self.paint_last_frame = False
    self.paint_method = "Copy"
    self.permanent = False
    self.trail_sequence = []
    self.paint_last_frame = show_last_frame
```

```
        self.paint_method = painting_style

        if trail_length < 0:
            self.permanent = True
            self.background = np.zeros((frame_height,frame_width,3),dtype=np.uint8)
        if trail_length < 1:
            trail_length = 1
        for i in range(trail_length):
            self.trail_sequence.append(alfa_img.Alfa_img.sized_frame(frame_width, frame_height))

    def iterate(self):
        pass

    def update(self,new_frame):
        del self.trail_sequence[0]
        self.trail_sequence.append(new_frame)

    def paint_function(self, target_frame, color_normalizator, avg_gain):
        self.float_target = None
        if self.background.any():
            self.float_target = self.background.astype(np.float32)
        else:
            self.float_target = target_frame.astype(np.float32)
        self.float_target = self.iterate(self.float_target, self.paint_method, avg_gain)

        self.float_target = paint.normalize(self.float_target, color_normalizator) * 255
        target_frame = self.float_target.astype(np.uint8)

        if self.paint_last_frame:
            self.current_frame = self.trail_sequence[-1]
            self.mask_3ch = cv2.cvtColor(self.current_frame.mask, cv2.COLOR_GRAY2BGR)
            target_frame[self.mask_3ch > 0] = 0
            target_frame += self.current_frame.entire_img * (self.mask_3ch > 0)
        if self.permanent:
            self.background = self.float_target.astype(np.uint8)
        return target_frame
```

```
class standard_echo(echo.Echo):
    def __init__(self,trail_length, frame_width, frame_height,show_last_frame,painting_style):
        super().__init__(trail_length, frame_width, frame_height,show_last_frame,painting_style)

    def iterate(self, target_frame, paint_style, avg):
        for self.iterator in range(len(self.trail_sequence)):
            target_frame = paint.paint_mode(paint_style, target_frame, self.trail_sequence[self.iterator],
                                            self.iterator, len(self.trail_sequence), avg)
        return target_frame

class inverted_echo(echo.Echo):
    def __init__(self,trail_length, frame_width, frame_height,show_last_frame,painting_style):
        super().__init__(trail_length, frame_width, frame_height,show_last_frame,painting_style)

    def iterate(self, target_frame, paint_style, avg):
        self.counter = 0
        for self.iterator  in reversed(range(len(self.trail_sequence))):
            target_frame = paint.paint_mode(paint_style, target_frame, self.trail_sequence[self.iterator],
                                            self.counter, len(self.trail_sequence), avg)
            self.counter += 1
        return target_frame
```

- **klasy metod detekcji światła**

Zależnie od wybranego typu detekcji światła tworzona jest instancja klasy *MOG2_substractor, threshold_substractor* lub *no_substractor*. Każda z nich zawiera funkcję *generate_mask* która zwraca maskę pokrywającą obszary, w których następuje malowanie światłem. Klasy te zawierają też podstawowe zmienne charakterystyczne dla danej metody, w klasie *MOG2_substractor* znajduje się instancja klasy wbudowanej w *OpenCV* - *BackgroundSubstractor*, która wylicza model tła. W klasie *threshold_substractor* znajduje się wartość progu.

```
class MOG2_substractor():
    def __init__(self, in_video, frames_amount, var_minimum, var_maximum, var_thres):
        self.MOG2 = cv2.createBackgroundSubtractorMOG2()
        self.MOG2.setVarMin(var_minimum)
        self.MOG2.setVarMax(var_maximum)
        self.MOG2.setVarThreshold(var_thres)
        self.frgrnd_mask = None
        for i in range(frames_amount):
            self.ret, self.background = in_video.read()
            self.frgrnd_mask = self.MOG2.apply(self.background)

    def generate_mask(self, frame):
        self.frgrnd_mask = self.MOG2.apply(frame, 0, 0)
        self.kernel = np.ones((3, 3), np.uint8)
        self.frgrnd_mask = cv2.erode(self.frgrnd_mask, self.kernel)
        self.frgrnd_mask = cv2.dilate(self.frgrnd_mask, self.kernel)
        self.ret, self.frgrnd_mask = cv2.threshold(self.frgrnd_mask, 0, 255, cv2.THRESH_OTSU)
        self.frgrnd_mask = cv2.medianBlur(self.frgrnd_mask, 9)
        return self.frgrnd_mask


class threshold_substractor():
    def __init__(self, thres_val):
        self.threshold = thres_val

    def generate_mask(self, frame):
        self.grayscale_img = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        self.ret, self.frame_mask = cv2.threshold(self.grayscale_img, self.threshold, 255,
cv2.THRESH_TOZERO)
        return self.frame_mask


class no_substractor():
    def generate_mask(self, frame):
        self.img_height, self.img_width, self.channels = frame.shape
        self.frgrd_mask = np.ones((self.img_height, self.img_width), dtype=np.uint8)
        return self.frgrd_mask
```

- **klasa *alfa_img***

Podstawowa klasa zawierająca dwie zmienne opisujące w pełni kolejną dodawaną do listy klatkę: obraz oraz odpowiadająca mu maska. Dodatkowo w klasie znajduje się funkcja aplikująca ręcznie maskę do obrazka, uwzględniając trójkanałowość obrazów. Dodatkowo dwie funkcje typu *@classmethod* pełniące funkcję dodatkowych konstruktorów.

```
class Alfa_img():
    def __init__(self, data_in, mask_in):
        self.entire_img = data_in
        self.mask = mask_in

    def apply_mask(self, destination):
        self.converted_data = self.entire_img
        self.mask_3ch = cv2.cvtColor(self.mask, cv2.COLOR_GRAY2BGR)
        destination[self.mask_3ch>0]=0
        destination += self.converted_data*(self.mask_3ch>0)
        return destination

    @classmethod
    def sized_frame(cls,frame_width,frame_height):
        data = np.zeros((frame_height,frame_width,3),dtype=np.uint8)
        mask = np.zeros((frame_height,frame_width),dtype=np.uint8)
        return cls(data, mask)

    @classmethod
    def one_frame(cls, frame_data):
        temp_data = frame_data
        temp_mask = None
        return cls(temp_data,temp_mask)
```

- **plik *paint.py***

W celu uproszczenia kodu w pliku głównym, stworzono dodatkowy plik, w którym umieszczono dwie funkcje:

  - *paint_mode*, funkcja realizująca sumowanie dwóch klatek zgodnie z wybranym wcześniej algorytmem
  - *normalize*, funkcja normalizująca, która zabezpiecza macierz obrazka przed przepełnieniem (sumując wartości pikseli często zachodzi sytuacja wyjścia poza standardowy zakres 0-1 dla typu float) i zwraca macierz, której wartości pikseli mieszczą się w wymaganym przedziale. Zależnie od wyboru użytkownika następuje albo prosta (i szybka obliczeniowo) normalizacja, czyli wartości kanałów poszczególnych pikseli, które przekroczą dopuszczalną wartość (1) są ustawiane odgórnie jako 1. Może to powodować przekłamania (np. gdy przepełni się jeden z kanałów RGB i zostanie on ustawiony na wartość 1, a pozostałe mają niższe wartości to przypisanie wartości maksymalnej 1 jednemu kanałowi zmieni stosunek wartości RGB i wrażenie barwy będzie inne). Aby temu zapobiec użytkownik może użyć „inteligentnej” normalizacji. Wtedy każdy piksel jest sprawdzany przez program niezależnie i w przypadku przepełnienia, każda z wartości RGB jest dzielona przez najwyższą wartość spośród trzech kanałów. Dzięki temu stosunek wartości kanałów RGB się nie zmienia. Wymaga to jednak zastosowania dwóch zagnieżdżonych pętli i sprawdzania każdego kanału w każdym pikselu niezależenie. Ze względu na to, że język *Python* nie jest zoptymalizowany do takich zadań to trwa to znacznie dłużej, jednak przynosi lepsze efekty.

```
def paint_mode(style, target_frame, frame_in, index, trail_size, average_multiplier):
    if style == "Copy":
        target_frame = frame_in.apply_mask(target_frame)
    elif style == "Accumulation":
        cv2.accumulate(frame_in.entire_img, target_frame, frame_in.mask)
    elif style == "Fading copy":
        weight = np.divide(index, trail_size)
        weighted_frame = np.multiply(frame_in.entire_img, weight)
        mask_3ch = cv2.cvtColor(frame_in.mask, cv2.COLOR_GRAY2BGR)
        target_frame[mask_3ch > 0] = 0
        target_frame += weighted_frame * (mask_3ch > 0)
    elif style == "Weighted accumulate (OpenCV function)":
        weight = np.divide(index, trail_size)
        cv2.accumulateWeighted(frame_in.entire_img, target_frame, weight, frame_in.mask)
    elif style == "Fading accumulation":
        weight = np.divide(index, trail_size)
        weighted_frame = np.multiply(frame_in.entire_img, weight)
        weighted_frame = weighted_frame.astype(np.float32)
        cv2.accumulate(weighted_frame, target_frame, frame_in.mask)
    elif style == "Average":
        weight = (float(1) / trail_size)+float(1 - float(1)/trail_size)*average_multiplier
        cv2.accumulateWeighted(frame_in.entire_img, target_frame, weight, frame_in.mask)
    return target_frame

def normalize(input_array,normalize):
    input_array = np.divide(input_array,255)
    max = np.amax(input_array)
    if max > 1:
        if normalize == True:
            max_pixel_val = 0
            array_height, array_width, num_channels = input_array.shape
            for i in range(array_height):
                for j in range(array_width):
                    max_pixel_val = np.amax(input_array[i,j])
                    if max_pixel_val > 1:
                        input_array[i,j,:] = np.divide(input_array[i,j,:],max_pixel_val)
        else:
            input_array[input_array > 1] = 1
    return input_array
```

- **moduł *window.py***

Ze względu na mnogość parametrów i wygodę obsługi oraz wyboru i zapisu plików zdecydowano się zastosować interfejs graficzny na bazie biblioteki *PyQt5*. Kod interfejsu graficznego znajduje się w plikach źródłowych programu.

- **przebieg programu**

Po wczytaniu wszystkich parametrów z interfejsu graficznego, otworzeniu źródłowego pliku video oraz utworzeniu instancji pliku wyjściowego video w formacie .mp4 z kodowaniem MPG4 następuje utworzenie właściwej (zależnej od wybranego parametru) instancji klasy pochodnej po klasie *Echo*:

```
if infinite_trail:
    trail_length = -1
else:
    trail_length = shadow_length
actual_time = time.strftime("%Y%m%d-%H%M%S")
in_video = cv2.VideoCapture(input_file)
if in_video.isOpened() == True:
    bad_file = False
    frame_height = int(in_video.get(4))
    frame_width = int(in_video.get(3))
    if source_fps == True:
        fps = int(in_video.get(cv2.CAP_PROP_FPS))
    else:
        fps = fps_val
    out_video = cv2.VideoWriter(output_file_dir+"/Light" + actual_time +".mp4",
                                cv2.VideoWriter_fourcc('M','P','G','4'), fps, (frame_width,frame_height))

    if inverted_trail:
        actual_trail_sequence = trail_type.inverted_echo(trail_length, frame_width,
                                                         frame_height,paint_last_frame,painting_method)
    else:
        actual_trail_sequence = trail_type.standard_echo(trail_length, frame_width,
frame_height,paint_last_frame,painting_method)
```

Następnie zależnie od wybranego trybu detekcji światła następuje utworzenie instancji klasy *MOG2_substractor, threshold_substractor* lub *no_substractor*. W przypadku wyboru algorytmu MOG2, następuje pobranie określonej liczby początkowych klatek z pliku i „nauka" tła wewnątrz konstruktora klasy. Algorytm MOG2 umożliwia użytkownikowi wybór czy tło ma być pokazane w pliku wyjściowym czy też usuwane i zastępowane czernią:

```
 if substraction_method == "Background substraction":
    substraction = background_substraction.MOG2_substractor(in_video, background_learn_frames,
                                                             var_min, var_max, var_threshold)
    if show_background:
        bckgd = np.zeros((frame_height,frame_width,3),dtype=np.uint8)
        bckgd = cv2.BackgroundSubtractor.getBackgroundImage(substraction.MOG2)
        actual_trail_sequence.background = bckgd
elif substraction_method == "Thresholding":
    substraction = background_substraction.threshold_substractor(threshold_level)
elif substraction_method == "Simple sum":
    substraction = background_substraction.no_substractor()
```

Następnie wykonywana jest pętla programu, aż do końca pliku video, w której pobierane są klatki i do każdej z nich generowana jest maska w sposób opisany wcześniej. W przypadku określonej, skończonej liczby *Trail length* każde przejście pętli powoduje uaktualnienie zestawu klatek w liscie (w przypadku nieskończonego śladu, lista ma długość 1 i każda kolejna klatka jest nakładana na sumę wszystkich poprzednich i zapisywana jako tło – oczywiście z uwzględnieniem maski). Następnie wywoływana jest funkcja „malująca" na podstawie wybranego trybu i wszystkich klatek w liście:

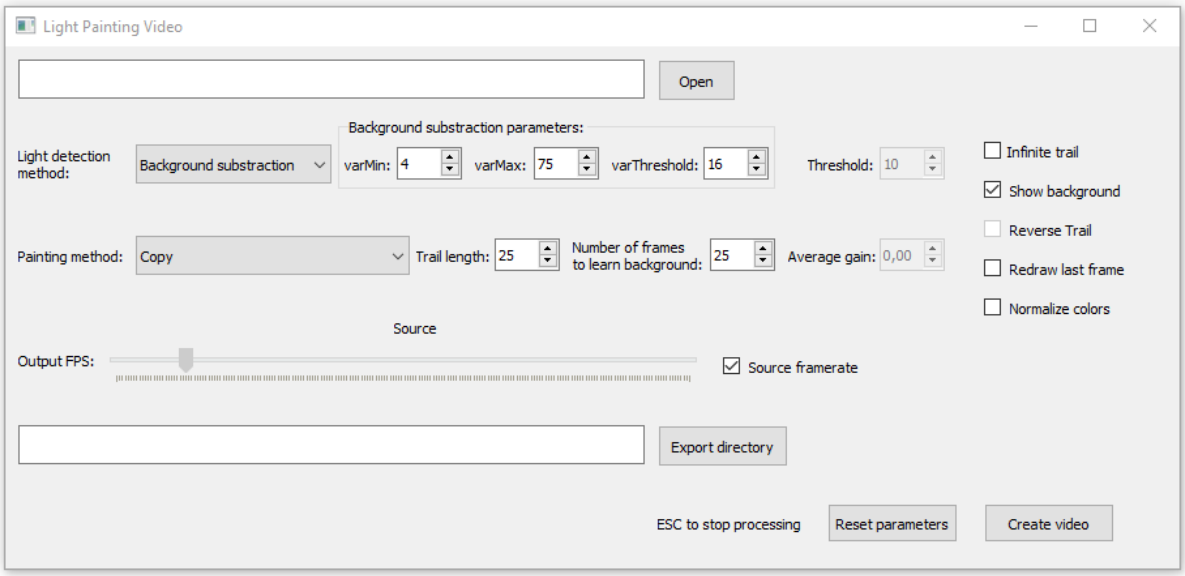
```
while(in_video.isOpened()):
    ret, input_frame =  in_video.read()
    if ret==True:
        foreground = alfa_img.Alfa_img(input_frame, substraction.generate_mask(input_frame))
        actual_trail_sequence.update(foreground)
```

```
    final_img = np.zeros_like(input_frame).astype(np.uint8)
    final_img = actual_trail_sequence.paint_function(final_img, color_normal, avrg_gain)
    out_video.write(final_img)
    cv2.imshow("Test",final_img)
    if cv2.waitKey(1) == 27:
        break
else:
    break
```

Na końcu programu następuje zamknięcie wszystkich okien podglądu oraz ewentualne zwrócenie informacji o wybraniu błędnego pliku do odczytu co powoduje potem wyświetlenie graficznego komunikatu w oknie.

5. Interfejs graficzny i opis parametrów

Po uruchomieniu programu pojawia się okno jak na rys.1.



Rysunek 1 Okno programu

Program pozwala na wygodne wprowadzenie wszystkich parametrów i szybką ich zmianę w przypadku niezadowalających efektów, ponieważ podczas przetwarzania video efekty działania programu są widoczne na bieżąco w dodatkowym oknie. Przerwanie przetwarzania przed jego zakończeniem jest możliwe poprzez wciśnięcie klawisza ESC. Dzięki temu wygodnie można modyfikować parametry przetwarzania i widzieć efekty już po kilku przetworzonych klatkach.

Parametry możliwe do ustawienia przez użytkownika:

- **Open -** wybór pliku video do przetworzenia, program przyjmuje pliki video akceptowalne przez bibliotekę *OpenCV*
- **Light detection method** – wybór metody detekcji światła, metody zostały opisane powyżej

- **Background substraction parameters: varMin, varMax, varThreshold** – są to parametry charakterystyczne tylko dla funkcji *BackgroundSubstractor*, w dokumentacji *OpenCV* nie zostały dokładnie opisane, natomiast ich wartości mogą się wahać w granicach 0-200. Mają bardzo duży wpływ na dopasowanie masek do rzeczywistych obiektów na materiale video. Określają wariancję jaką może charakteryzować się obiekt/tło aby można było zakwalifikować dane piksele do odpowiedniej grupy. Większe wartości są odpowiednie dla obiektów bardziej kontrastowych (np. jasna lampa na czarnym tle).
- **Threshold** – próg z jakim mają być oddzielane obiekty maskowane od tła, tylko dla metody *Thresholding*. Wartość parametru w zakresie 0-255.
- **Painting method** – wybór metody kopiowania, metody zostały opisane w poprzednich punktach
- **Taill length** – długość śladu liczona w klatkach – im więcej tym dłuższy czas obliczeń
- **Number of frames to learn background** – parametr określający ile początkowych klatek materiału video będzie wykorzystane tylko do „uczenia" się tła. Charakteryzuje tylko metodę *Background substraction*
- **Average gain** – opisany wcześniej parametr wzmacniający intensywność pikseli klatek wykorzystywanych do sumowania w metodzie *Average*
- **Output FPS** – ile klatek na sekundę ma mieć wyjściowy materiał video, domyślnie zaznaczone, że wartość ma być taka sama jak w materiale wejściowym. Ustawienie mniejszej liczby klatek spowoduje spowolnienie filmu a większej przyspieszenie.
- **Source framerate** – ustawia domyślną liczbę klatek na sekundę taką jak materiału wejściowego
- **Export directory** – wybór folderu docelowego do zapisania materiału
- **Infinite trail** – określa czy ślad ma być nieskończony (czyli zapisywany od początku materiału do końca)
- **Show background** – określa czy tło ma być wyświetlane, czy nie (dotyczy tylko *Background substraction*)
- **Reverse trail** – określa kierunek zanikania śladu w przypadku jego ograniczonej długości i trybów *Fading copy, Weighted accumulate (OpenCV function), Fading copy*. Zaznaczona opcja powoduje, że ślad zanika przy źródle światła.
- **Redraw last frame** – określa czy najnowsza klatka ma być narysowana w pełnej jasności (widoczne w przypadku trybu *Averag*e gdzie poszczególne klatki mają niską jasność lub w przypadku śladu zanikającego w kierunku odwrotnym)
- **Normalize colors** – aktywuje albo deaktywuje „inteligentną" normalizację opisaną powyżej. Zaznaczenie znacząco wydłuża czas przetwarzania.
- **Reset parameters** – umożliwia zresetowanie wszystkich wartości (oprócz ścieżek dostępu) do wartości domyślnych
- **Create video** – uruchamia przetwarzanie

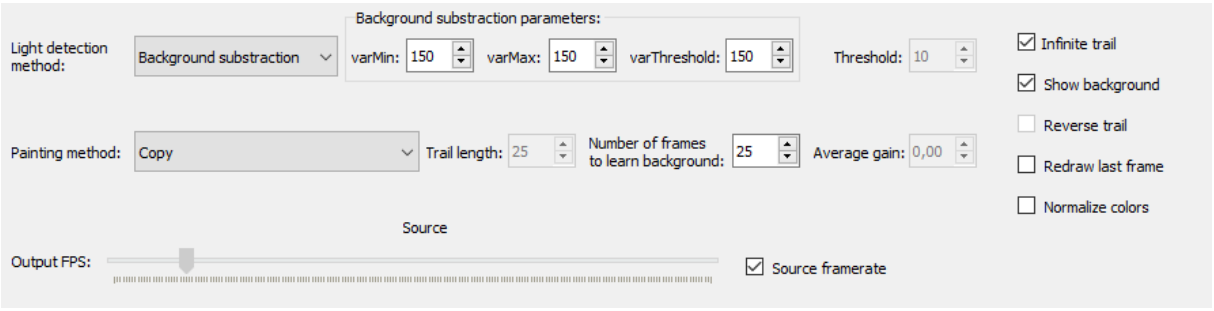
Program ma zaimplementowaną najbardziej podstawową obsługę błędów tzn. opcje które nie są dostępne dla danego trybu są „wyszarzone" w interfejsie graficznym (np. dla trybu *Average* nie ma możliwości ustawienia nieskończonego śladu gdyż nie byłoby możliwe wyliczenie wagi każdej klatki) albo dla trybu *Copy* nie można zaznaczyć *Reverse trail* gdyż tryb ten nie oferuje „zanikającego" śladu, przy którym odwrócone zanikanie jest widoczne.

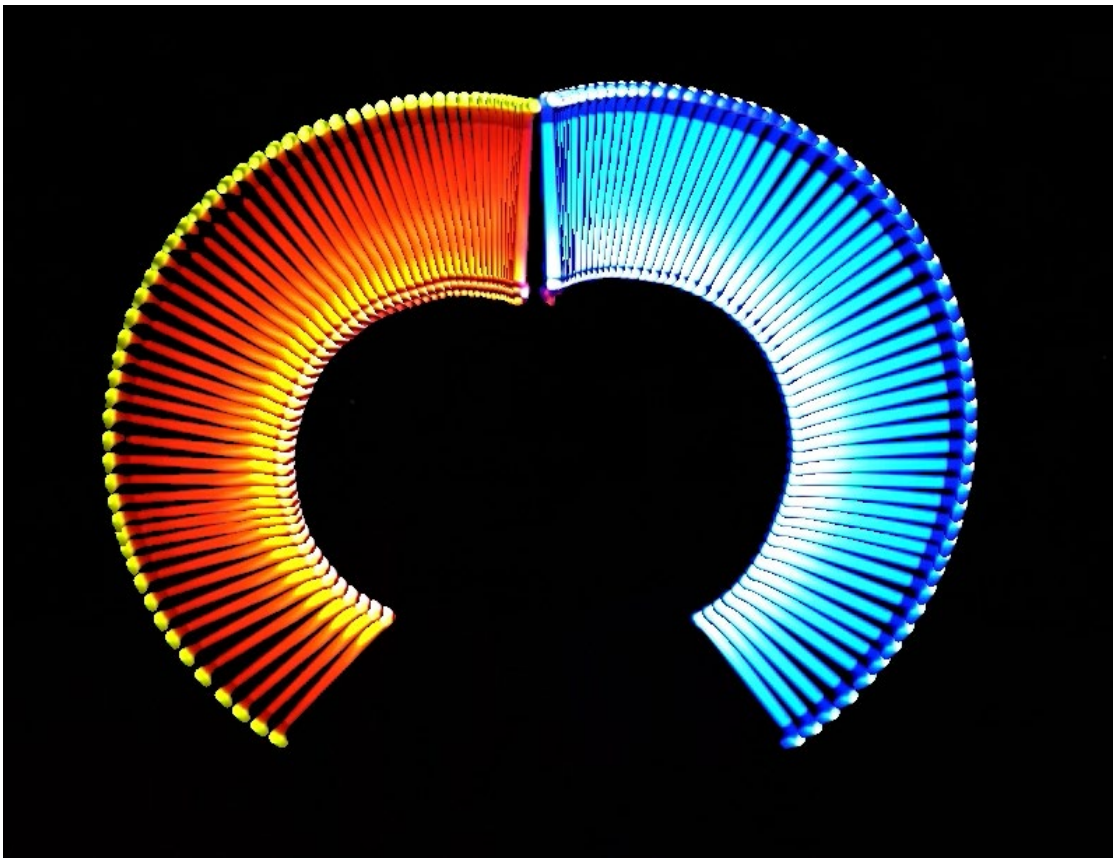
Wskazanie błędnego pliku lub brak ścieżek dostępu również spowoduje wyświetlenie komunikatu o błędzie.

6. Efekty działania

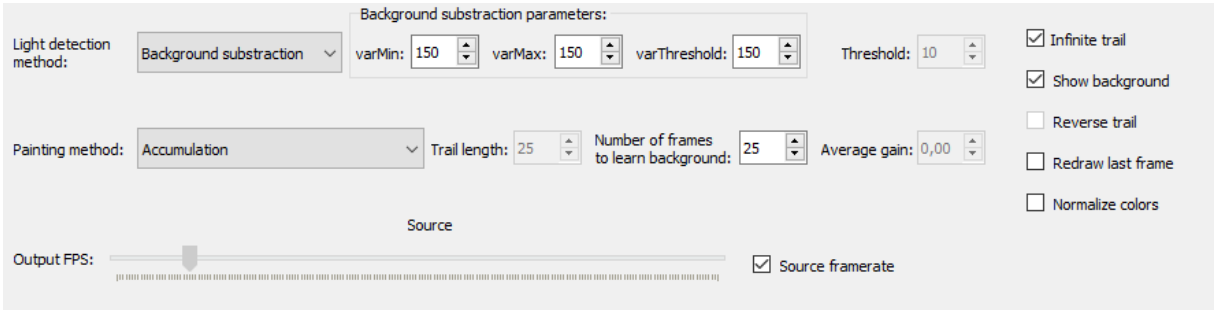
Ze względu na mnogość parametrów, liczba efektów do uzyskania z jednego materiału video jest nieskończona i niemożliwa do pełnego zaprezentowania. W materiałach dodatkowych przedstawiono przykładowy filmik źródłowy wraz z przykładowymi efektami po przetworzeniu przez program. Rzeczywisty efekt działania programu można zilustrować tylko na materiale video, na stopklatce wygląda to jak standardowe malowanie światłem w fotografii. Bardzo duży wpływ na efekt końcowy ma liczba klatek na sekundę materiału wejściowego. Im więcej klatek tym bardziej ciągła i płynna jest smuga światła. Poniżej przedstawiono kilka przykładowych kadrów dla jednego materiału źródłowego (10 FPS), ale różnych parametrów programu:

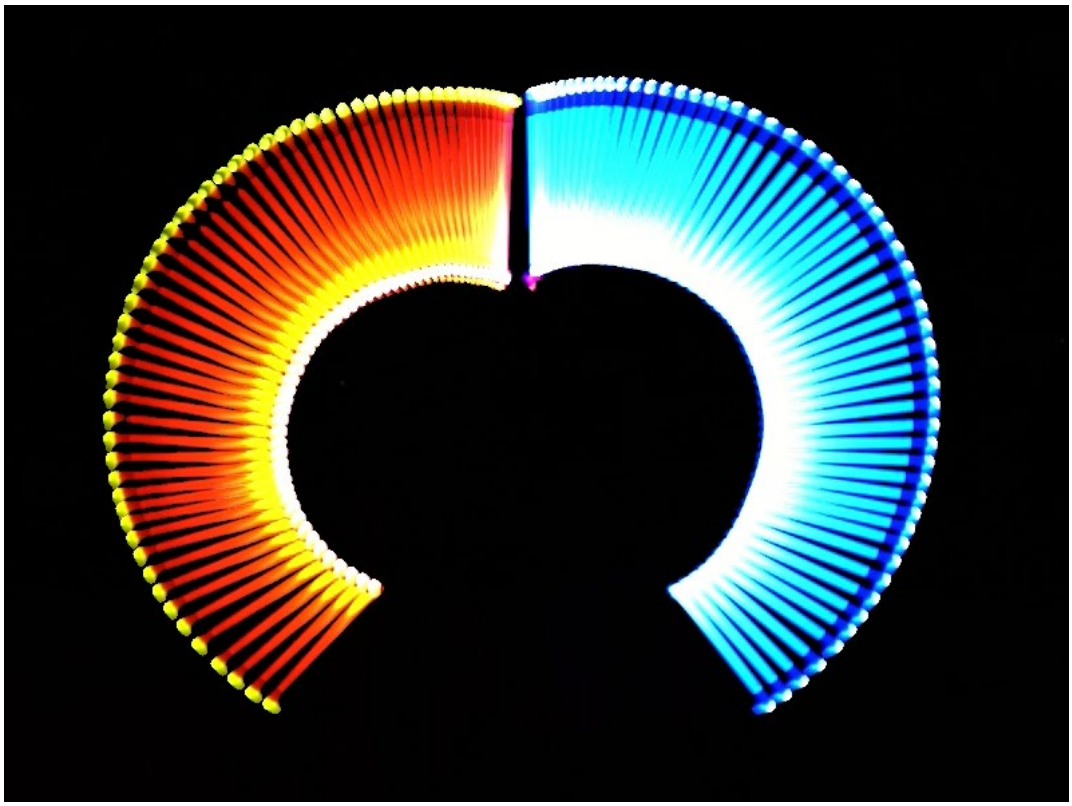
1.



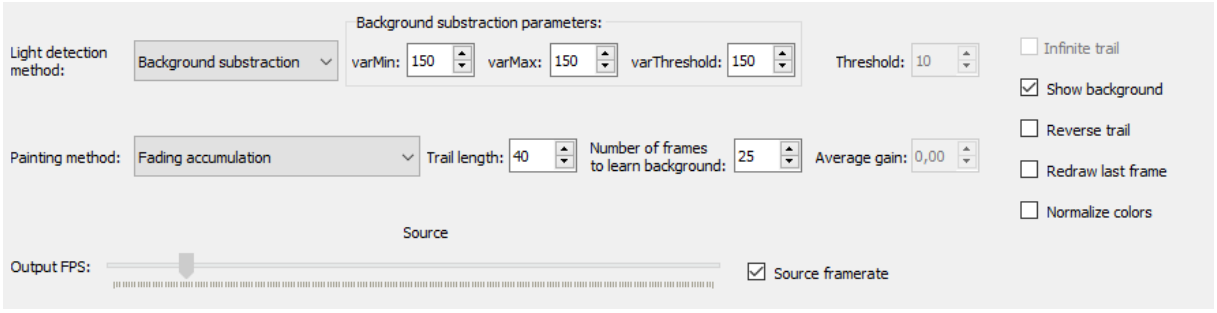


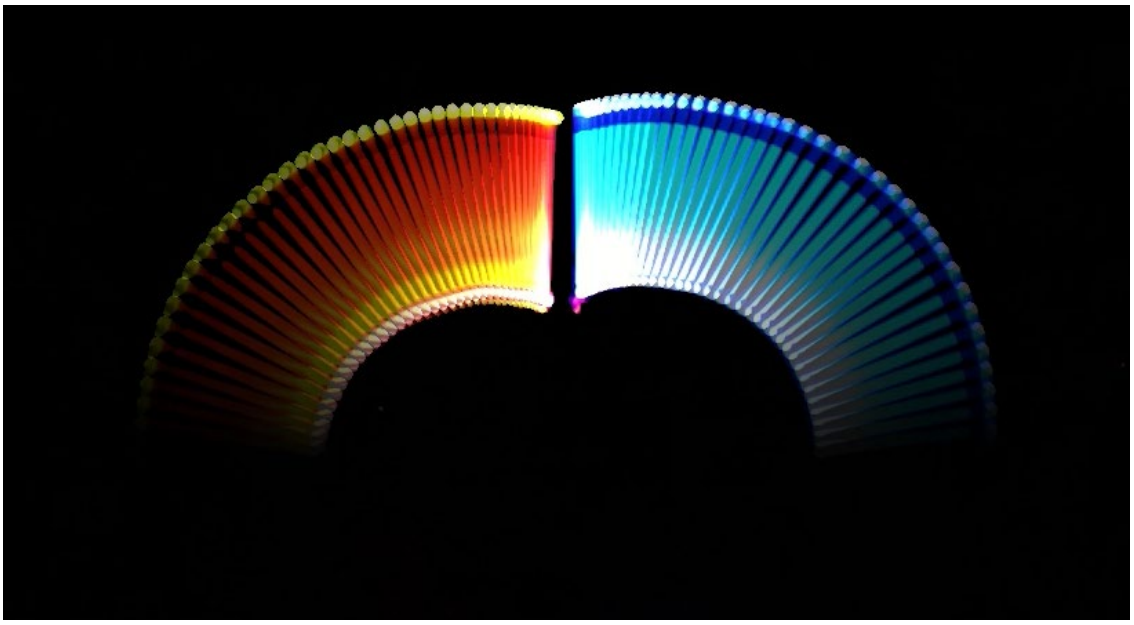
2.


Background substraction parameters:
Light detection method:
Background substraction
varMin: 150
varMax: 150
varThreshold: 150
Threshold: 10
Infinite trail
Show background
Reverse trail
Redraw last frame
Normalize colors
Painting method:
Accumulation
Trail length: 25
Number of frames to learn background: 25
Average gain: 0,00
Source
Output FPS:
Source framerate

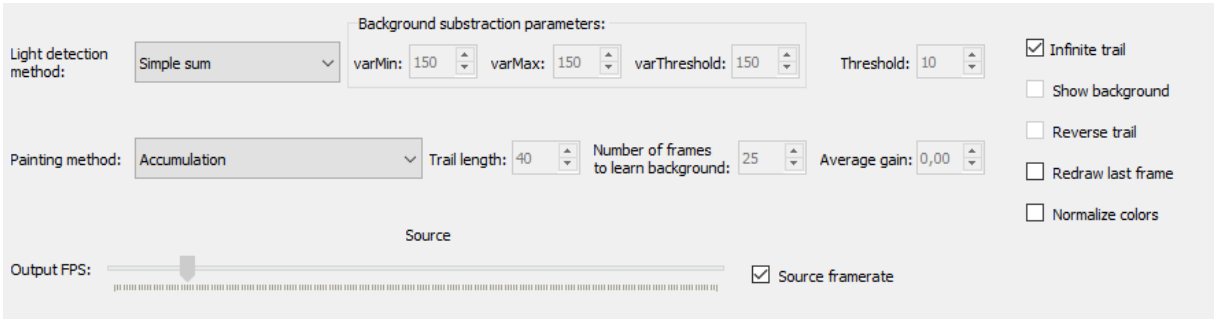


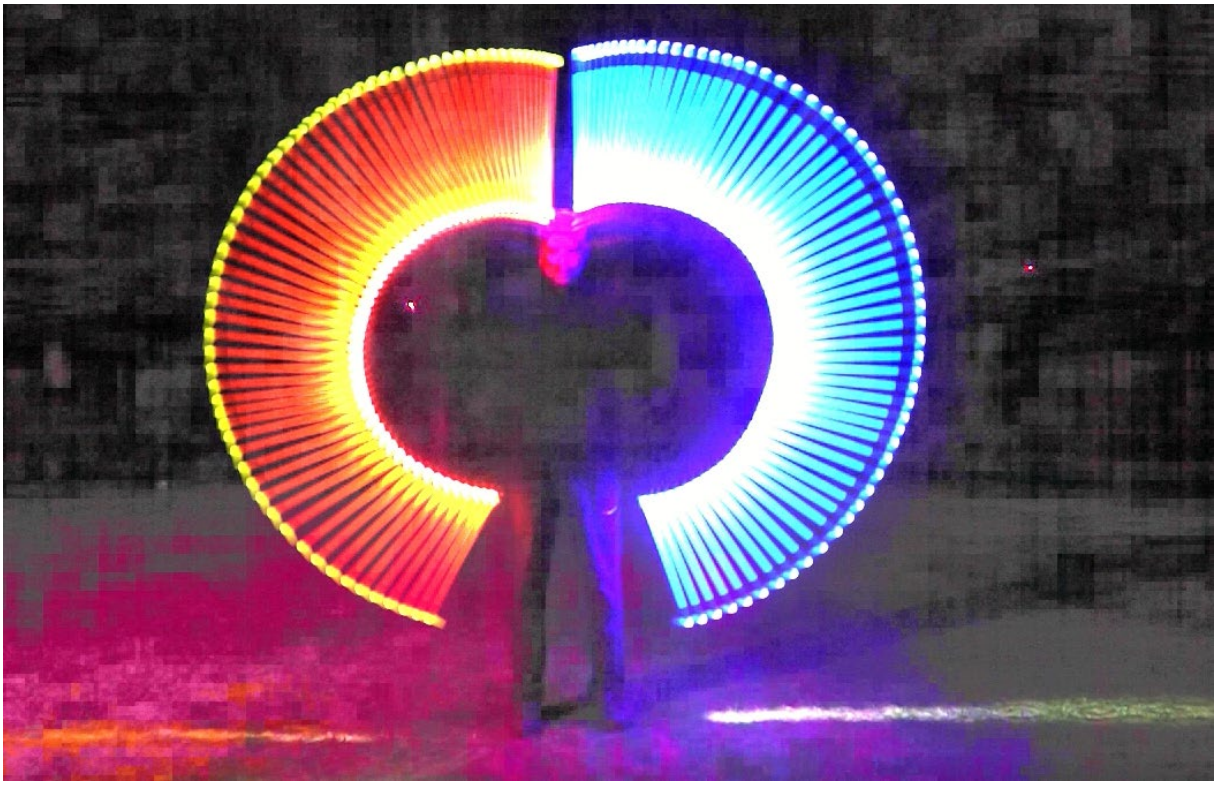
3.


Background substraction parameters:
Light detection method:
Background substraction
varMin: 150
varMax: 150
varThreshold: 150
Threshold: 10
Infinite trail
Show background
Reverse trail
Redraw last frame
Normalize colors
Painting method:
Fading accumulation
Trail length: 40
Number of frames to learn background: 25
Average gain: 0,00
Source
Output FPS:
Source framerate

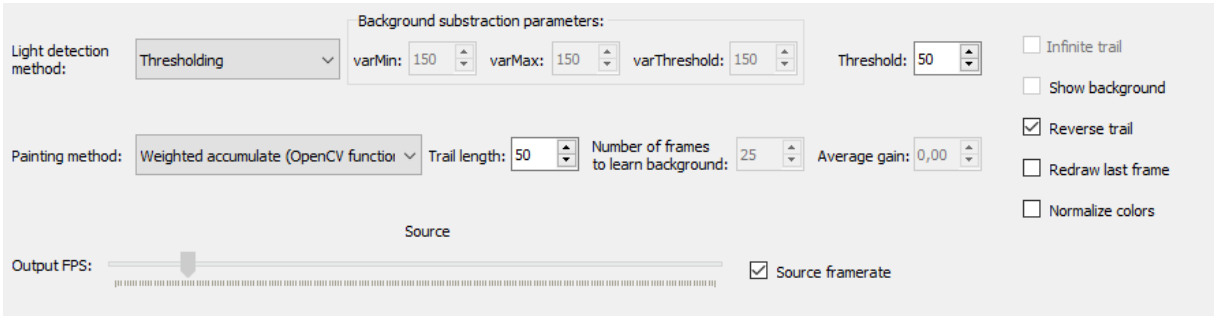


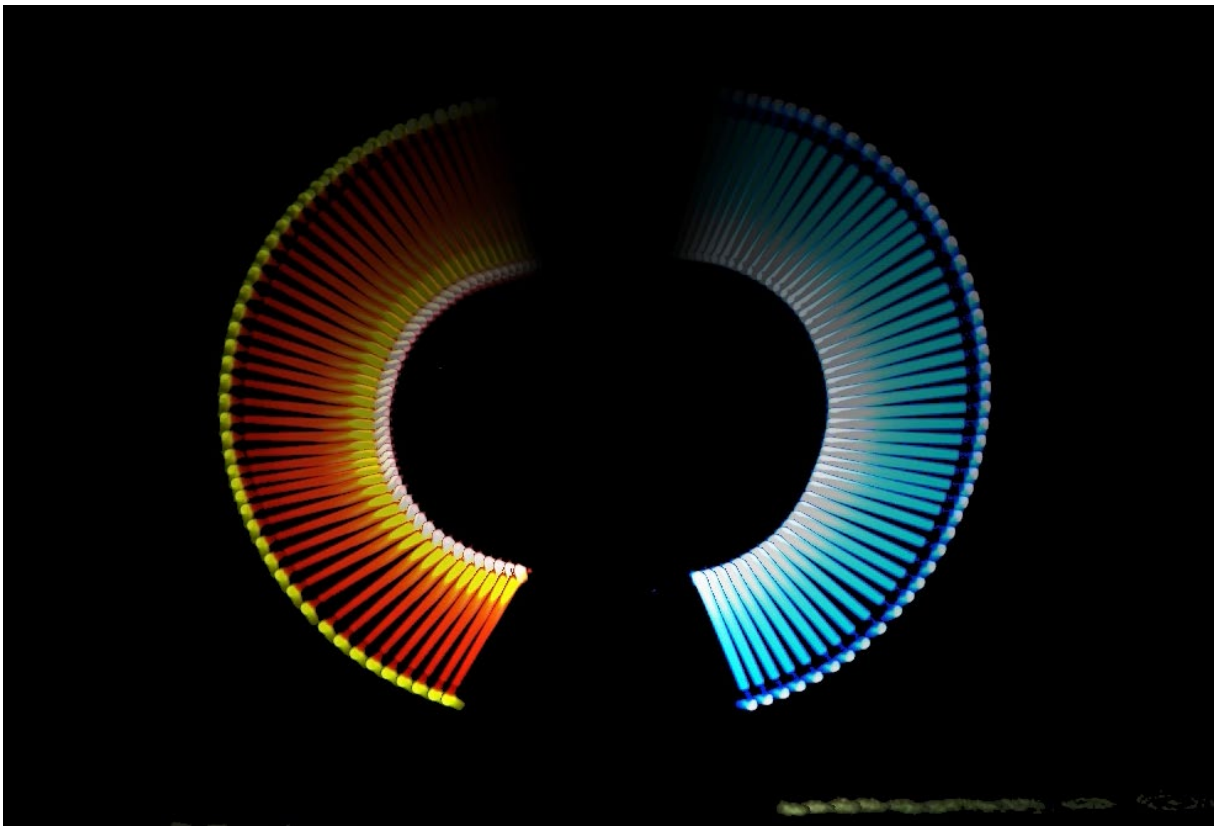
4.


Background substraction parameters:
Light detection method:
Simple sum
varMin: 150
varMax: 150
varThreshold: 150
Threshold: 10
Infinite trail
Show background
Reverse trail
Painting method:
Accumulation
Trail length: 40
Number of frames to learn background: 25
Average gain: 0,00
Redraw last frame
Normalize colors
Source
Output FPS:
Source framerate

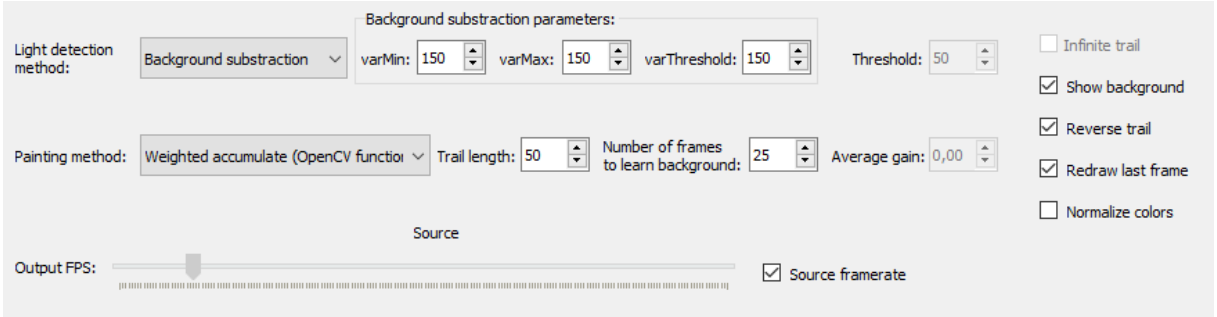


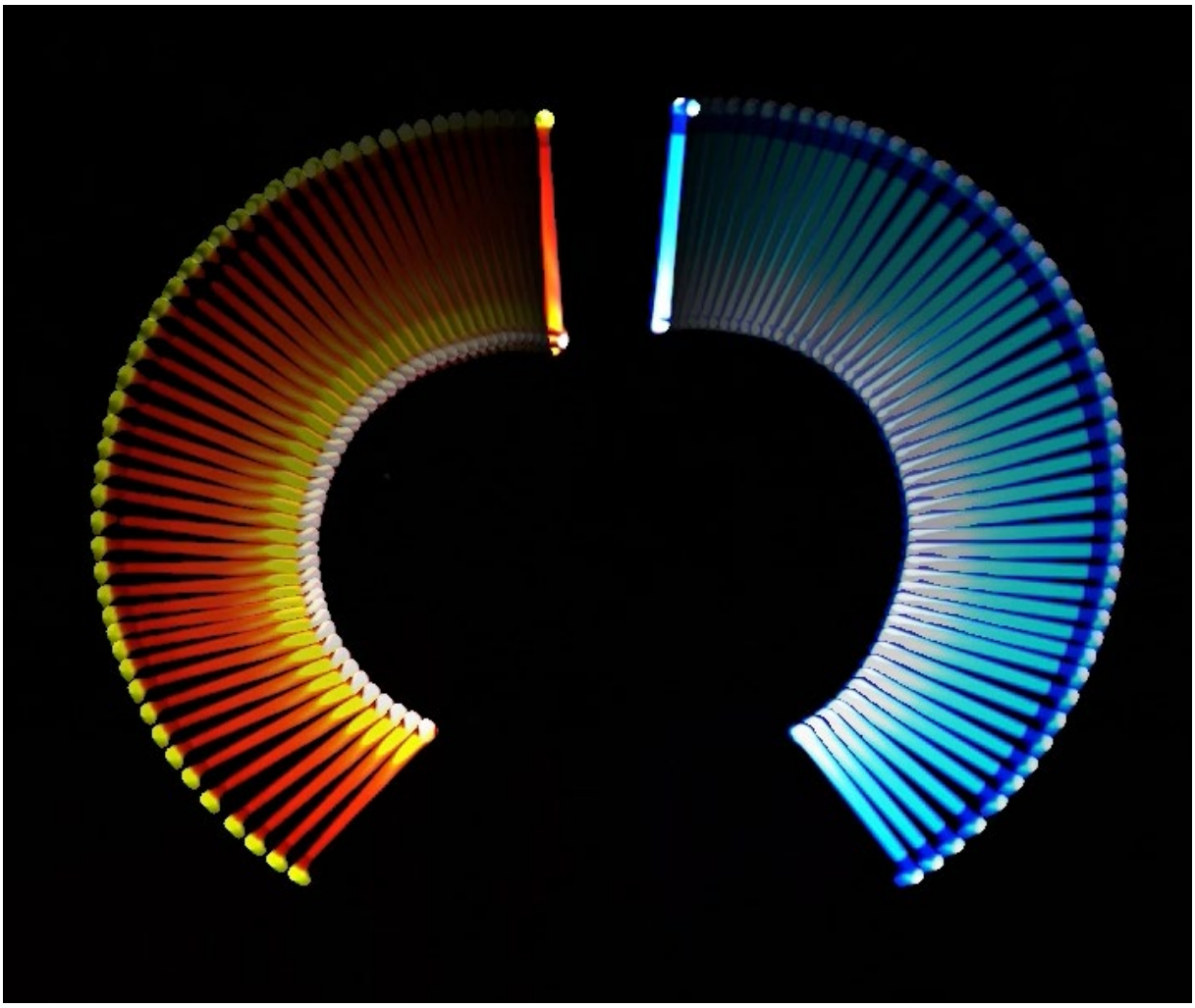
5.


Background substraction parameters:
Light detection method:
Thresholding
varMin: 150
varMax: 150
varThreshold: 150
Threshold: 50
Infinite trail
Show background
Reverse trail
Painting method:
Weighted accumulate (OpenCV functio
Trail length: 50
Number of frames to learn background: 25
Average gain: 0,00
Redraw last frame
Normalize colors
Source
Output FPS:
Source framerate

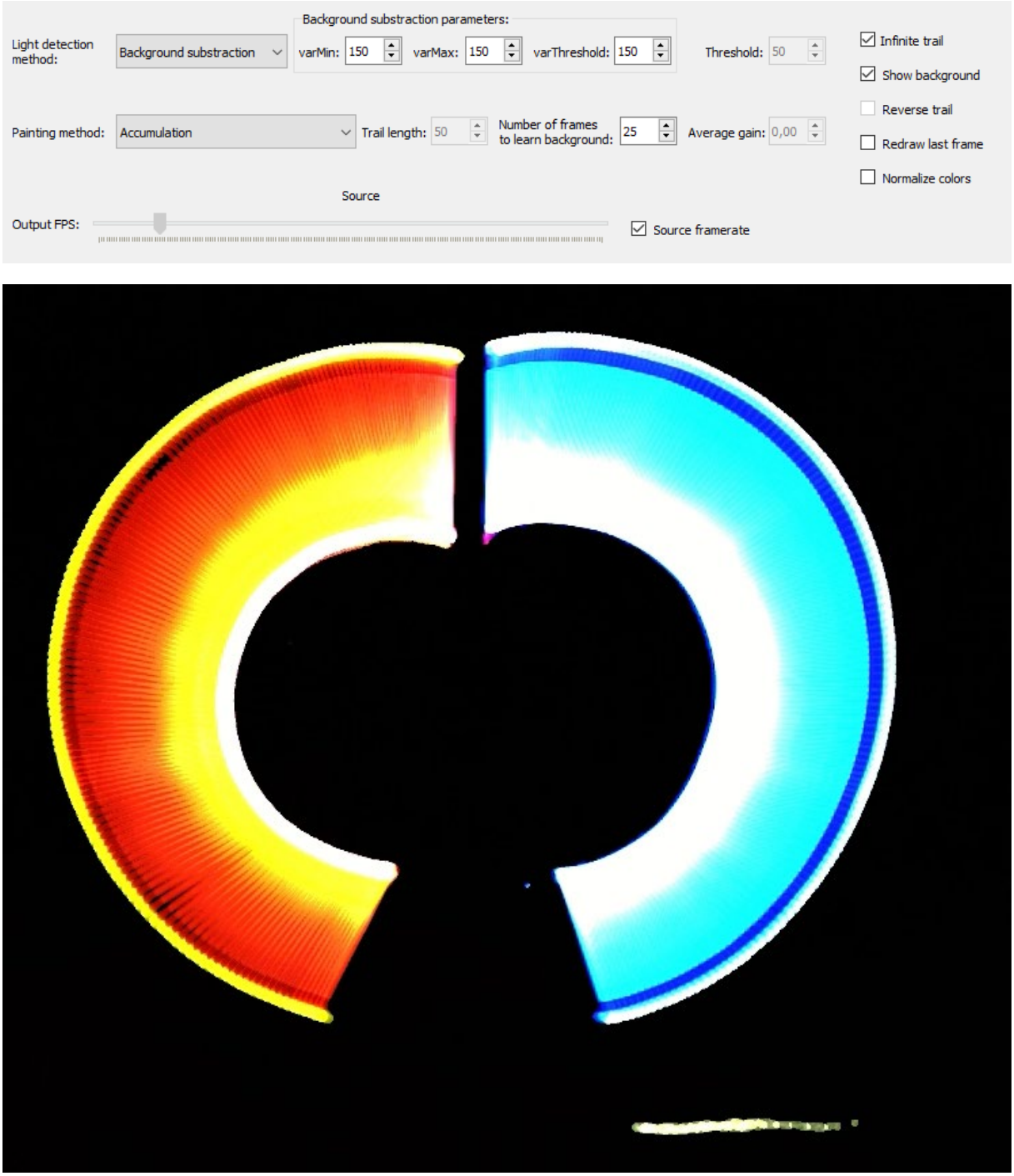


6.


Background substraction parameters:
Light detection method:
Background substraction
varMin: 150
varMax: 150
varThreshold: 150
Threshold: 50
Infinite trail
Show background
Reverse trail
Painting method:
Weighted accumulate (OpenCV functio
Trail length: 50
Number of frames to learn background: 25
Average gain: 0,00
Redraw last frame
Normalize colors
Source
Output FPS:
Source framerate



Powyższe efekty uzyskano dla jednego materiału o liczbie klatek na sekundę równej 10, poniżej klatka wynikowa dla tego samego materiału, ale w wersji 25 FPS, parametry takie same jak dla przykładu numer 2 powyżej:



7. Podsumowanie

Udało się stworzyć program spełniający założenia projektowe. Ze względu na to, że niemożliwe jest określenie uniwersalnych parametrów dających satysfakcjonujące rezultaty dla każdego wejściowego video, to zdecydowano się rozbudować program i umożliwić użytkownikowi ustawienie możliwie wielu parametrów zależnie od potrzeb.