Efekt Sepii
Języki Asemblerowe

...

Autor:
Michał Jankowski,
Informatyka
studia stacjonarne,
stopień I, sem.V, gr.I, sekcja II,
Rok akademicki 2019/2020

Spis treści

Założenia

- Przekształcenie danego obrazu na odcienie szarości
- Dodanie współczynnika wypełnienia W dla zadanych składowych piksela do wcześniej przekształconego obraz w odcienie szarości
- Wykonanie GUI w Windows Forms w języku C#
- Biblioteka napisana w C#
- Biblioteka odpowiedzialna za przekształcanie obrazu w odcienie szarości

Opis algorytmu

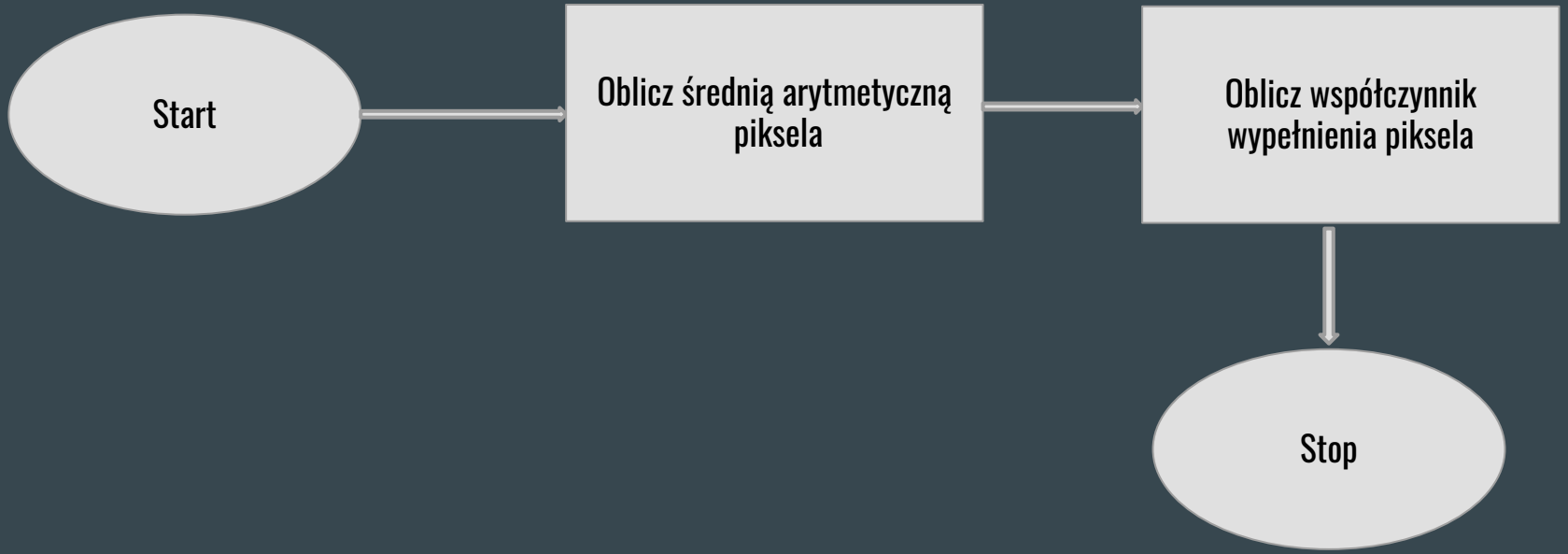
Jedną ze znanych metod uzyskania efektu sepii jest przekształcenie kolorowego obrazu na odcienie szarości, a następnie dokonanie koloryzacji tego obrazu zadanym współczynnikiem wypełnienia W.

$$R = (R + G + B) / 3$$
$$G = (R + G + B) / 3$$
$$B = (R + G + B) / 3$$

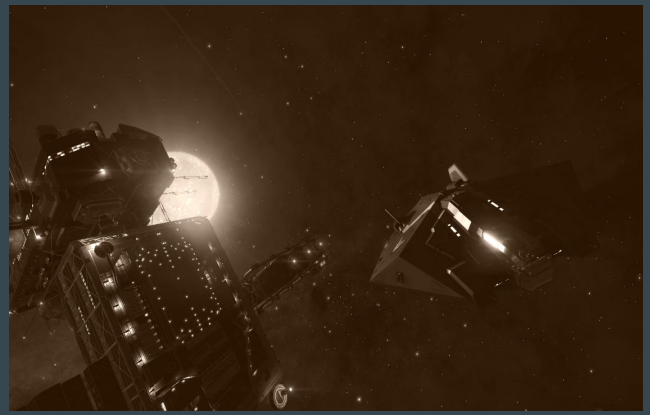
$$R = R + 2 * W$$
$$G = G + W$$
$$B = B$$

R - war. składowej czerwonej
G - war. składowej zielonej
B - war. składowej niebieskiej

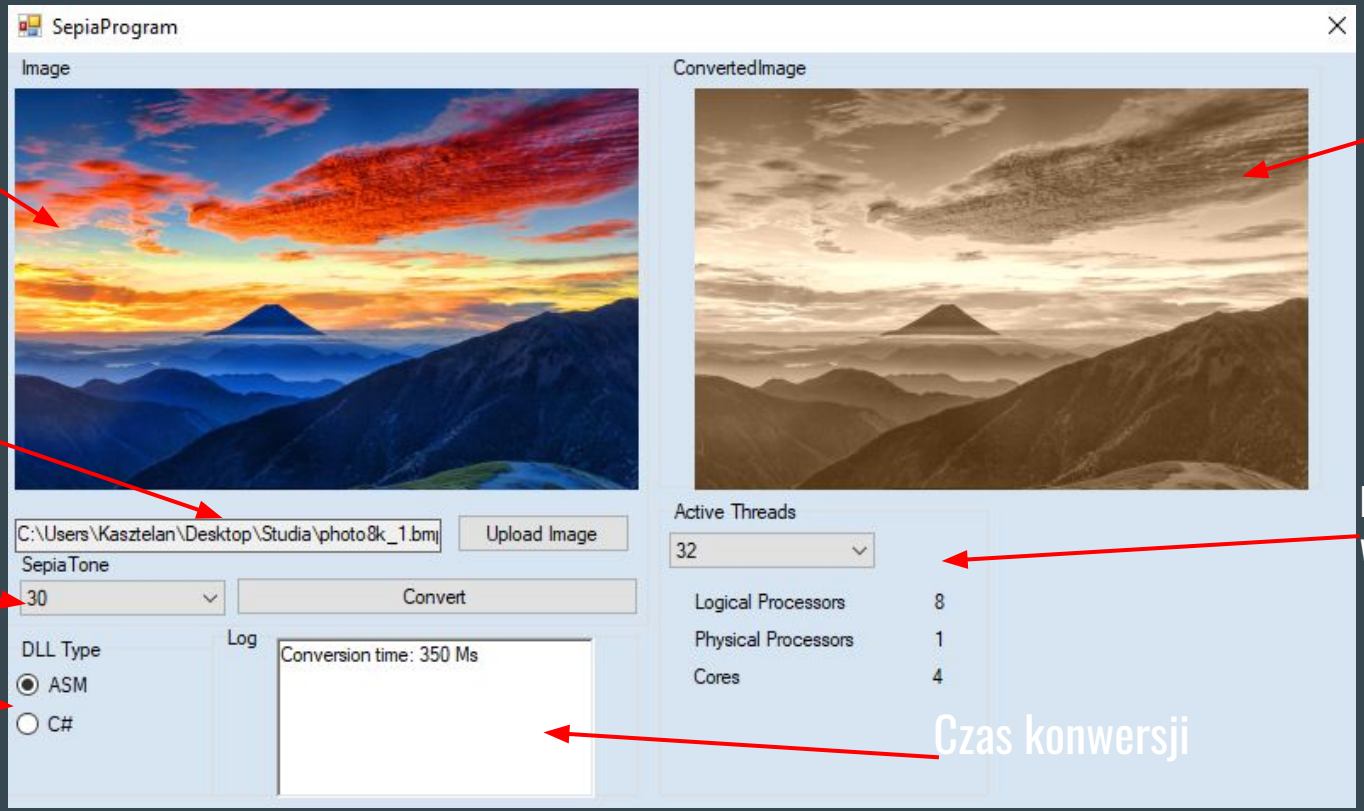
Schemat blokowy



Zasada działania programu



Budowa programu



Żądany Obraz

Skonwertowany Obraz

Lista Przeszukiwań

Wypełnienie Sepią

Typ Dll

Liczba rdzenii i wątków

Czas konwersji

Kod w C#

```
namespace SepiaDll
{
    //Klasa obsługująca efekt Sepii dla C#
    0 references
    public class Sepia
    {
        0 references
        public static void  CSharpDllFunc(object args)
        {
            //tworzenie tablicy 4 obeiktów typu Array
            Array arguments = new object[4];
            //Rzutowanie danego obiektu na Array
            arguments = (Array)args;

            //Początek przedziału tablcy bajtów dla danego waątku
            int RGBstart = (int)arguments.GetValue(0);
            //Koniec przedziału tablicy bajtów dla danego wątku
            int RGBstop = (int)arguments.GetValue(1);
            //tablica bajtów dla danego wątku
            byte[] rgbValues = (byte[])arguments.GetValue(2);
            //wartość efektu sepii
            int sepiaValue = (int)arguments.GetValue(3);
```

```
//Początek obługi konwersji obrazka, otrzymanie efektu czarości
for (int i = RGBstart; i <= RGBstop - 4; i += 4)
{
    //tworzenie zmiennej average i zapisanie danych tablicy i-tej do tej zmiennej
    int average = rgbValues[i];
    //dodanie wartości i+1 tablicy do average
    average += rgbValues[i + 1];
    //dodanie wartości i+2 tablicu do average
    average += rgbValues[i + 2];
    //dzielenie wartości average przez 3, aby otrzymać średnią
    average /= 3;
    // wpisywanie jako bajt wartości średniej do i-tej wartości tablicy
    rgbValues[i] = (byte)average;
    // wpisywanie jako bajt wartości średniej do i+1 wartości tablicy
    rgbValues[i + 1] = (byte)average;
    // wpisywanie jako bajt wartości średniej do i+2 wartości tablicy
    rgbValues[i + 2] = (byte)average;

}
```

Kod w C#

```
//Pętla tworząca efekt sepii dla przetworzonego obrazka odcienie czarości
for (int i = RGBstart; i <= RGBstop - 4; i += 4)
{
    //warunek sprawdzający czy przekroczono wartość 255 dla koloru zielonego
    if ((rgbValues[i + 1] + sepiaValue) > 255)
        //Ustawienie maksymalnej dozwolonej wartości koloru
        rgbValues[i + 1] = 255;
    else
        rgbValues[i + 1] = (byte)(rgbValues[i + 1] + sepiaValue);
    //warunek sprawdzający czy przekroczono wartość 255 dla koloru czerwonego
    if ((rgbValues[i + 2] + 2 * sepiaValue) > 255)
    //Ustawienie maksymalnej dozwolonej wartości koloru
        rgbValues[i + 2] = 255;
    else
        rgbValues[i + 2] = (byte)(rgbValues[i + 2] + 2 * sepiaValue);

}
```

Fragment kodu w ASM

```
average_loop:                          ; start pętli wykonującej odcienie szarości
movdqu xmm10, [rdi]                    ; pobranie 2 pikseli z rejestru rdi do xmm10

add rdi, 16                             ; przesuniecie rejestru indeksowego o 16 pozycji do przodu, w celu pobrania nowych wartości z rejestru rdi
sub rcx, 16                             ; przesuniecie rejestru zliczaczjqvego o 16 pozycji do tyłu, aby nie wyjść poza rejestr

movdqu xmm9, [rdi]                      ;  pobranie kolejnych 2 pikseli z rejestru rdi do xmm9


vinsertf128 ymm4,ymm4,xmm10,0          ; przesuniecie 2 pikseli do gornej czesci rejestru ymm4
vinsertf128 ymm4,ymm4,xmm9,1           ; przsuniecie 2 piskeli do dolnej czesci rejestru ymm4
                                       ; przetrzymywanie akutalnie 4 pikseli jednocześnie w rejestrze ymm4


vmovaps ymm6, ymm4                     ; zapamietanie skladowych alpha 4 kolejnych pikseli w rejestrze ymm6


vpand ymm6, ymm6, ymm2                  ; maskowanie kanału przezroczystoći 4 kolejnych pikseli w rejestrze ymm6

vmovaps ymm7,ymm4                      ; przepisanie 8 kolejnych pikseli bez kanału przezroczystości do rejestru ymm7

vpslldq ymm7,  ymm7 , 1                ; logiczne przesunięcie rejestru ymm7 o 1 w lewo
vmovaps ymm8, ymm7                     ; przesunięcie rejstru ymm7 do ymm8
vpslldq  ymm8,ymm8, 1                  ; logiczne przesunięcie rejestru ymm8 o 1 w lewo

vpand ymm4, ymm4, ymm0                 ; zamaskowanie składowych a, g, b rejestru ymm4 z wykorzystaniem maski kolory czerwonego z rejestru ymm0. Uzyskujemy same kolory czerwone R
vpand ymm7,ymm7,ymm0                   ; zamaskowanie składowych a, r, b rejestru ymm7 z wykorzystaniem maski kolory czerwonego z rejestru ymm0. Uzyskujemy same kolory zielony G
vpand ymm8, ymm8, ymm0                 ; zamaskowanie składowych a, r, b rejestru ymm8 z wykorzystaniem maski kolory czerwonego z rejestru ymm0. Uzyskujemy same kolory zielony B

vpaddd ymm5, ymm4, ymm7                ; zsumowanie rejestrów ymm4 i ymm7 do rejestru ymm5. Uzyskujemy sume kolory czerwonego R i zielonego G. R+G
vpaddd ymm5,ymm5, ymm8                 ; zsumowanie rejestrów ymm5 i ymm8 do rejestru ymm5. Uzyskujemy sume koloru R+G oraz koloru niebieskiego B. R+G+B


                                       ; dzielenie wartosci 3 kolejnych skladowcyh pikseli przez wartosc 3
vcvtdq2ps ymm7, ymm5                   ; zamiana wartości  int rejestru ymm5 na float do rejestru ymm7 w celu wykonania dzielenia wektorowego
vdivps ymm7, ymm7, ymm3                ; dzielenie wektorowe rejestru  ymm7 przez rejestr stałego dzienika 3 do rejestru ymm7
vcvtps2dq ymm5,ymm7                    ; zamiana wartości float rejestru ymm7 na int do rejstru ymm5.
```

Zastosowanie wątków

- Liczba wątków od 1 do 64
- Każdy wątek oblicza dany fragment obrazu
- Podział na początek i koniec przedziału
- Wątki otrzymują fragmenty tablicy bajtów

Przygotowanie wątków

```
private  void createThreads()
{
    //brak tablicy wątków lub tablica jest pusta
    if (arrayOfThreads == null)
        return;
    //początek zliczania czasu
    model.startWatch();
    for (int i = 0; i< arrayOfThreads.GetLength(0); ++i)
    {
        //Start wątku
        arrayOfThreads[i].Start(arrayOfArguments[i]);

    }
}
```

```
private void waitForThreads()
{
    //tablica wątków pusta
    if (arrayOfThreads == null) return;

    bool done = false;
    //sprawdzenie czy wątek się zakończył
    while (!done)
    {
        done = true;

        for (int i = 0; i < arrayOfThreads.GetLength(0); ++i)
            // wartość bool sprawdzająca czy tablica wątków jest pusta lub czy dany wątek "nie żyje"
            done &= (arrayOfThreads[i] == null || !arrayOfThreads[i].IsAlive);
    }
}
```

```
public void PrepareDelegates(RadioButton csharp,RadioButton asm, int threads, int toneValue)
{
    //nie przekazano żadnej tablicy albo tablica jest pusta
    if (RGBValuesOfImage == null)
        return;
    // wątek 0 (Visualowy) dla  Asm
    if (appThreadAsm(asm, threads, toneValue) == true)
        return;
    // wątek 0 (Visualowy) dla C#
    if (appThreadCsharp(csharp, threads, toneValue) == true)
        return;
    arrayOfThreads = new Thread[threads];
    arrayOfArguments = new object[threads];
    //podział tablice na wątki
    int interval = RGBValuesOfImage.GetLength(0) / threads;
   // pętla tworząca interwały
    for(int i = 0; i< arrayOfThreads.GetLength(0); ++i)
    {
        int start = i * interval;
        //uzupełenienie do pełnego piksela dla początku przedziału

        while(start %4 != 0)
        {
            start += 1;
        }
        int stop = (i + 1) * interval;
        //uzupełenienie do pełnego piksela dla końcu przedziału
        while (stop %4 != 0)
        {
            stop += 1;
        }
        //tablica obiektów przekazywana dla każdego wątku z odpowiednią wartością sepii
        arrayOfArguments[i] = new object[4] { start, stop, RGBValuesOfImage, toneValue };
        //nowy wątek dal C#
        if (csharp.Checked) arrayOfThreads[i] = new Thread(new ParameterizedThreadStart(SepiaDll.Sepia.CSharpDllFunc));
        //nowy wątek dla asmeblera
        else if (asm.Checked) arrayOfThreads[i] = new Thread(new ParameterizedThreadStart(assemblerFunction));

    }
}
```

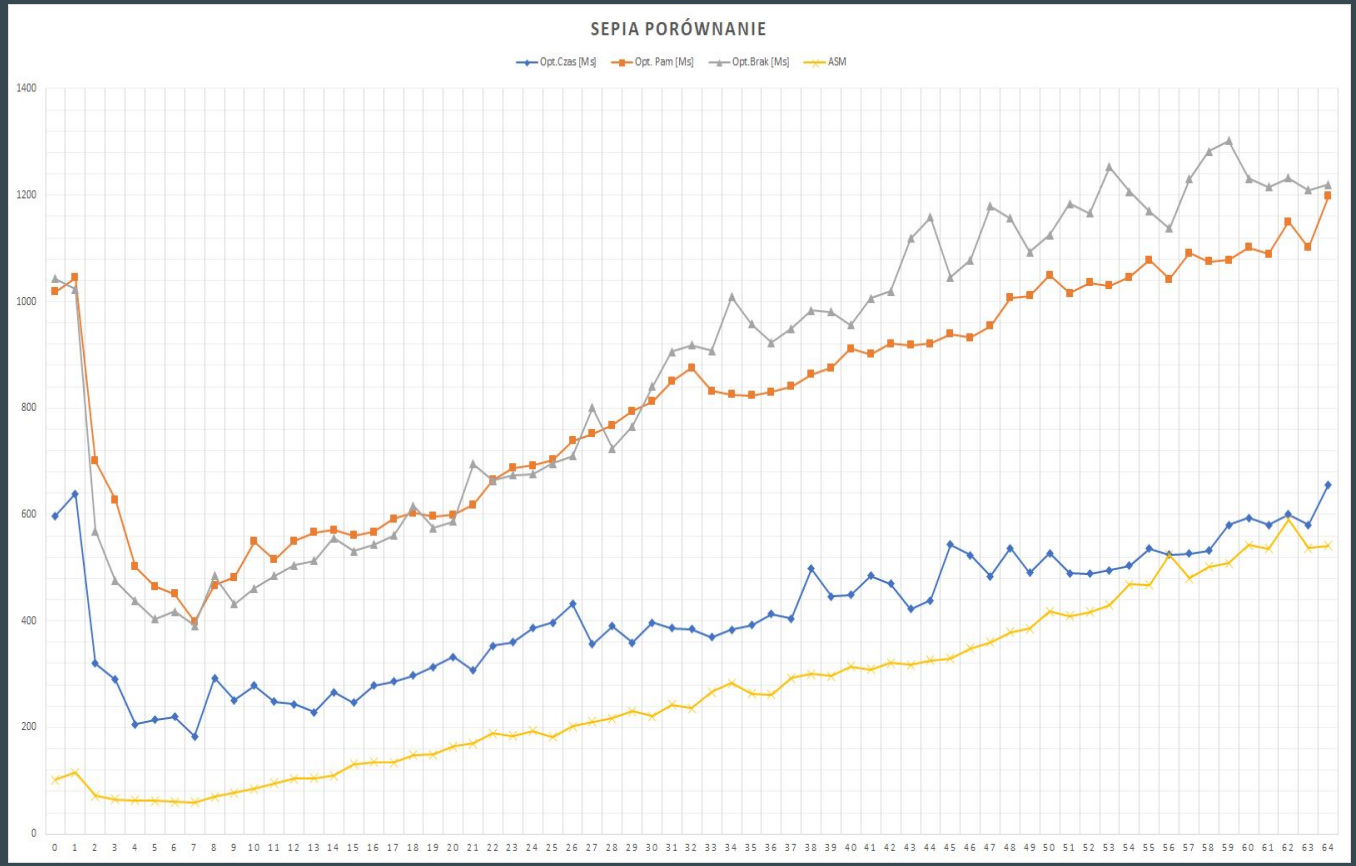
Sprawdzanie zgodności

- Obsługa operacji SSE za pomocą rejestru EDX
- Obsługa operacji AVX za pomocą rejestru ECX
- Kontrola wymaganej wersji frameworka

Testowanie

Wnioski:

- ASM okazał się optymalny, ale dla liczby wątków większej niż 57 C# po optymalizacji czasowej uzyskał porównywalne wyniki
- Optymalizacja pamięciowa ~~(śr. 20 KB na 7 KB dll)~~ została uzyskana kosztem szybkości wykonania algorytmu
- Dla wątków 1 i 2 odnotowano największą różnice w wykonaniu algorytmu



Podsumowanie i wnioski

- Użycie instrukcji wektorowych
- Skomplikowana implementacja
- Szybkość wykonywanych operacji w asemblerze
- Konieczność sprawdzania dostępnych instrukcji
- Możliwości obecnych kompilatorów