Języki asemblerowe

ROZMYCIE GAUSSA

Agenda

- Czym jest rozmycie Gaussa?
- Przykłady rozmycia Gaussa
- Analiza algorytmu
- Interfejs użytkownika
- Implementacja
- Porównanie czasów
- Wnioski

Czym jest Rozmycie Gaussa?

- Rozmycie Gaussa (znane również jako wygładzenie gaussowskie) to powszechnie używana funkcja oprogramowania graficznego wykorzystywana w celu zmniejszenia szumów i zakłóceń w obrazie lub w celu zamazania detali. Rozmycie Gaussa bardzo często jest wykorzystywane w profesjonalnej obróbce zdjęć. Wygładzanie gaussowskie jest również stosowane jako etap wstępnego przetwarzania obrazów w wizji komputerowej.

Przykład rozmycia Gaussa

Przed rozmyciem

Po rozmyciu

Przykład rozmycia Gaussa

Przed rozmyciem

Po rozmyciu

Działanie filtru w rozmyciu Gaussa

- Przetworzenie obrazu wiąże się ze zmianą wartości jego pikseli aby otrzymany efekt był widoczny dla oka. Każdy piksel w obrazie posiada swoją wagę, to za pomocą niej jesteśmy w stanie przekształcić obraz. Piksele przetwarzane są przez maskę. Maski filtrujące obraz mają różne rozmiary: 3x3, 5x5 lub 7x7. Rozmiar maski którą spotyka się najczęściej wynosi 3x3. Program stworzony przeze mnie również wykorzystuje maskę 3x3.

Analiza algorytmu

- Proces filtracji składa się z przetwarzania każdej składowej obrazu osobno. W przypadku modelu RGB należy przeprowadzić osobno filtrację dla składowych R, G oraz B. Podczas filtracji pikseli znajdujących się na krawędziach obrazu dochodzi do sytuacji gdy maska filtrująca wychodzi poza przetwarzany obraz. Istnieje kilka sposobów aby poradzić sobie z problemem. Jednym z rozwiązań jest zmniejszenie zakresu obejmowania maski na krawędziach obrazu.

Analiza algorytmu – przedstawienie maski

Szkielet wybranej maski

$f_{-1,-1}$	$f_{0,-1}$	$f_{1,-1}$
$f_{-1,0}$	$f_{0,0}$	$f_{1,0}$
$f_{-1,1}$	$f_{0,1}$	$f_{1,1}$

Wagi pól wybranej maski dla rozmycia Gaussa

1	2	1
2	4	2
1	2	1

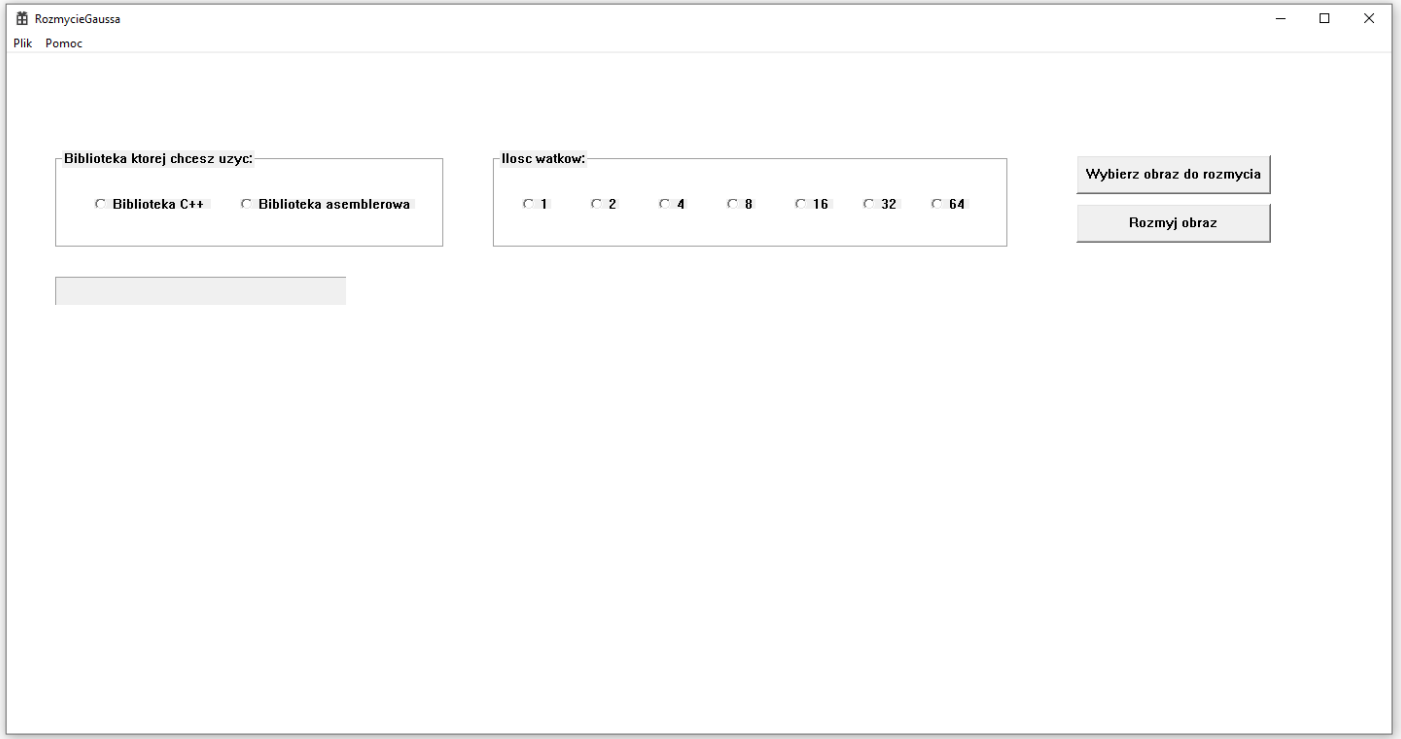
Analiza algorytmu – obliczanie składowych

Obliczamy nową wartość składowej punktu a o współrzędnych (i,j) według następującego wzoru:

- Obliczamy sumę ważoną składowej punktu i wszystkich sąsiadów zgodnie z wagami wskazanymi przez maskę filtra:
- $s = f_{-1,-1} * a_{i-1,j-1} + f_{0,-1} * a_{i,j-1} + f_{1,-1} * a_{i+1,j-1} + f_{-1,0} * a_{i-1,j} + f_{0,0} * a_{i,j} + f_{1,0} * a_{i+1,j} + f_{-1,1} * a_{i-1,j+1} + f_{0,1} * a_{i,j+1} + f_{1,1} * a_{i+1,j+1}$

- Sumę dzielimy przez sumę wszystkich wag maski:
- $a'_{i,j} = \frac{s}{f_{-1,-1}+f_{0,-1}+f_{1,-1}+f_{-1,0}+f_{0,0}+f_{1,0}+f_{-1,1}+f_{0,1}+f_{1,1}}$

Interfejs użytkownika



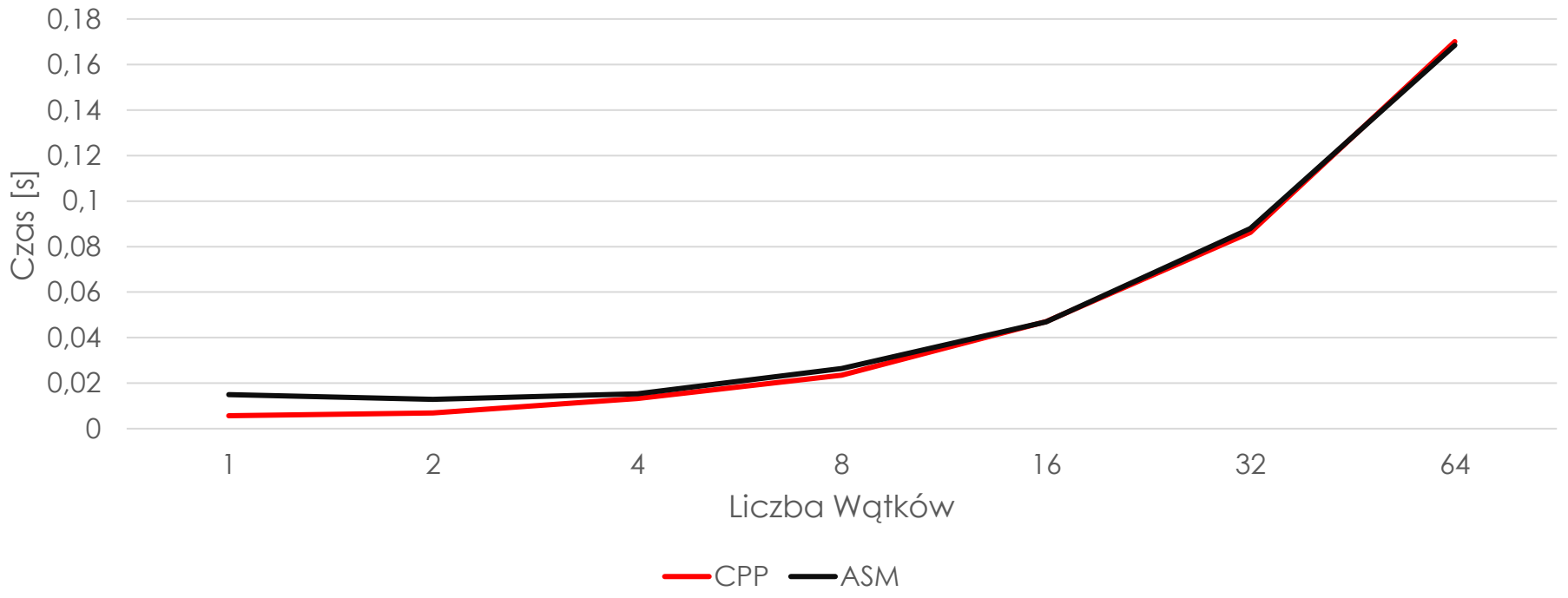
Implementacja

```
loopOverMaskRows:                ;for (int j = -1; j <= 1; j++)
comisd xmm7, xmm8                ;Sprawdź czy j < 2
je loopOverMaskRowsDone          ;Jeżeli warunek pętli loopOverMaskRows nie jest spełniony wyjdź z niej

    movd xmm10, minusOne         ;Wyzerowanie rejestru zawierającego indeks petli wewnetrznej
    loopOverMaskColumns:         ;for (int i = -1; i <= 1; i++)
    comisd xmm10, xmm8           ;Sprawdź czy j < 2
    je loopOverMaskColumnsDone   ;Jeżeli warunek pętli loopOverMaskColumns nie jest spełniony wyjdź z niej

    ;Sprawdzenie poprawności ifa ;if ((row + j) >= 0 && (row + j) < height && (col + i) >= 0 && (col + i) < width)
    mov r11d, r9d                ;przypisanie do rejestru r11d aktualnego indeksu wiersza, gdyż row = r9d
    movd xmm4, r11               ;Wykorzystanie rejestru xmm4 jako bufor na wartość r11
    addsd xmm4, xmm7             ;operacja dodania do rejestru xmm4 wartości xmm7 => row + j, gdyż xmm7 = j
    movd r11, xmm4               ;przypisanie do rejestru r11 wartości xmm4, (row + j) -> r12
    movd r12, xmm0               ;przypisanie do rejestru r12 aktualnego indeksu kolumny, gdyż col = r8d
    movd xmm4, r12               ;Wykorzystanie rejestru xmm4 jako bufor na wartość r12
    addsd xmm4, xmm10            ;operacja dodania do rejestru xmm4 wartości xmm10 => col + i, gdyż xmm10 = i
    movd r12, xmm4               ;przypisanie do rejestru r12 wartości xmm4, (col + i) -> r12
    cmp r11d, 0                  ;Porównanie wartości rejestru r11d z zerem, porównanie (row + j) z 0
    jl isFalse                   ;jeżeli (row + j) < 0 przejdź do etykiety isFalse
    cmp r11d, r10d               ;Porównanie wartości rejestru r11d z r10d, porównanie (row + j) z endHight
    jge isFalse                  ;jeżeli (row + j) >= endHight przejdź do etykiety isFalse
    cmp r12d, 0                  ;Porównanie wartości rejestru r12d z zerem, porównanie (col + i) z 0
    jl isFalse                   ;jeżeli (col + i) < 0 przejdź do etykiety isFalse
    cmp r12d, r8d                ;Porównanie wartości rejestru r12d z r8d, porównanie (col + i) z width
    jge isFalse                  ;jeżeli (col + i) >= width przejdź do etykiety isFalse

    ;Jeżeli if jest spełniony
    ;Lewa strona wyrazenia w tablicy colorsBeforeFilter
    movd xmm11, rax;             ;Robie bufor z xmm11 na trzymanie rax, żeby wrócić potem do poprzedniej wartości
    mov eax, r11d                ;(row + j) -> eax
    mov r11d, 3                  ;Wpisuje 3 do rejestru r11d
    mul r11d                     ;(row + j) * 3 -> eax
    mul r8d                      ;(row + j) * 3 * width -> eax
    mov r11d, eax                ;Wpisuje wynik lewe strona wyrazenia w tablicy colorsBeforeFilter do rejestru r11d
    movd rax, xmm11              ;wracam rax do poprzedniej wartości

    ;Prawa strona wyrazenia w tablicy colorsBeforeFilter
    movd xmm11, rax;             ;Robie bufor z xmm11 na trzymanie rax, żeby wrócić potem do poprzedniej wartości
    mov eax, r12d                ;(column + i) -> eax
    mov r12d, 3                  ;Wpisuje 3 do rejestru r12d
    mul r12d                     ;(column + i) * 3 -> eax
    mov r12d, eax                ;Wpisuje wynik prawa strona wyrazenia w tablicy colorsBeforeFilter do rejestru r12d
    movd rax, xmm11              ;wracam eax do poprzedniej wartości
```

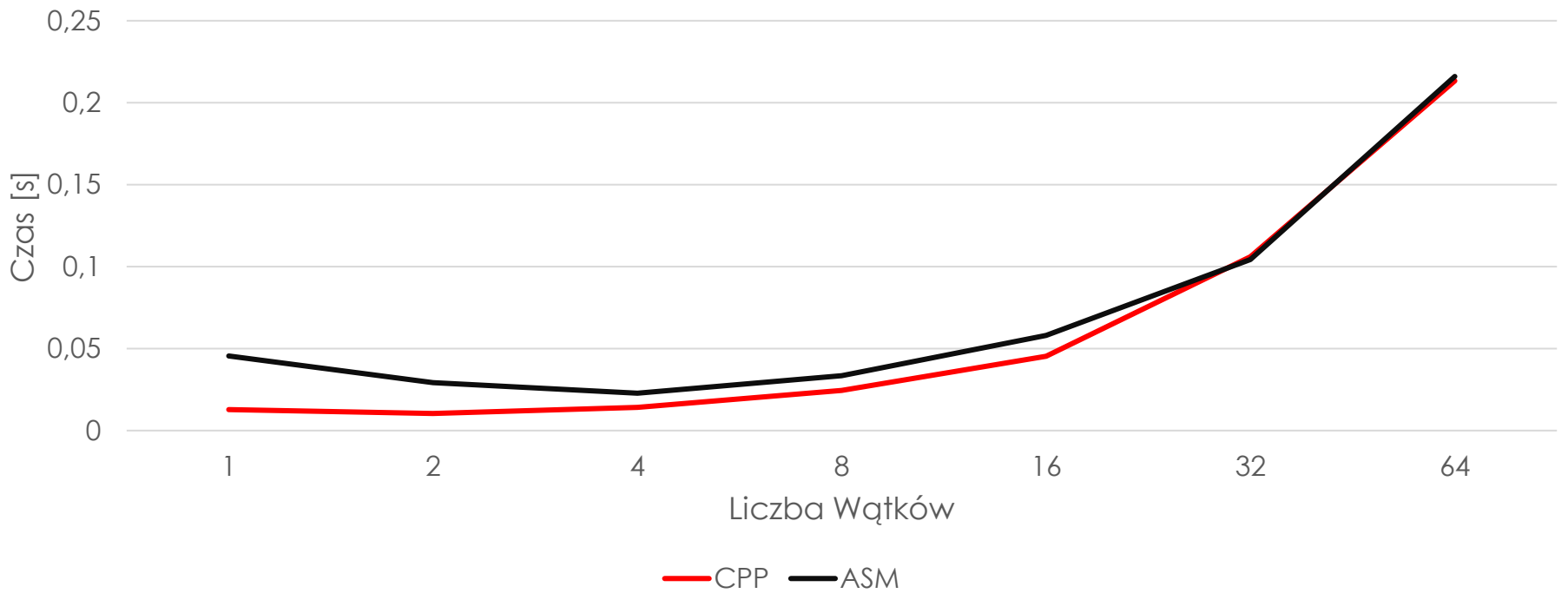
Porównanie czasów

- Wydajność programu sprawdzono przeprowadzając porównanie czasów filtracji dla obrazów o różnej wielkości. Rozdzielczości obrazów które analizowano są następujące:
- ✓ - 256x256
- ✓ - 640x426
- ✓ - 1280x1024
- ✓ - 1920x1080 (Full HD)
- ✓ - 3840x2160 (4K)
- ✓ - 7680x4320 (8K)

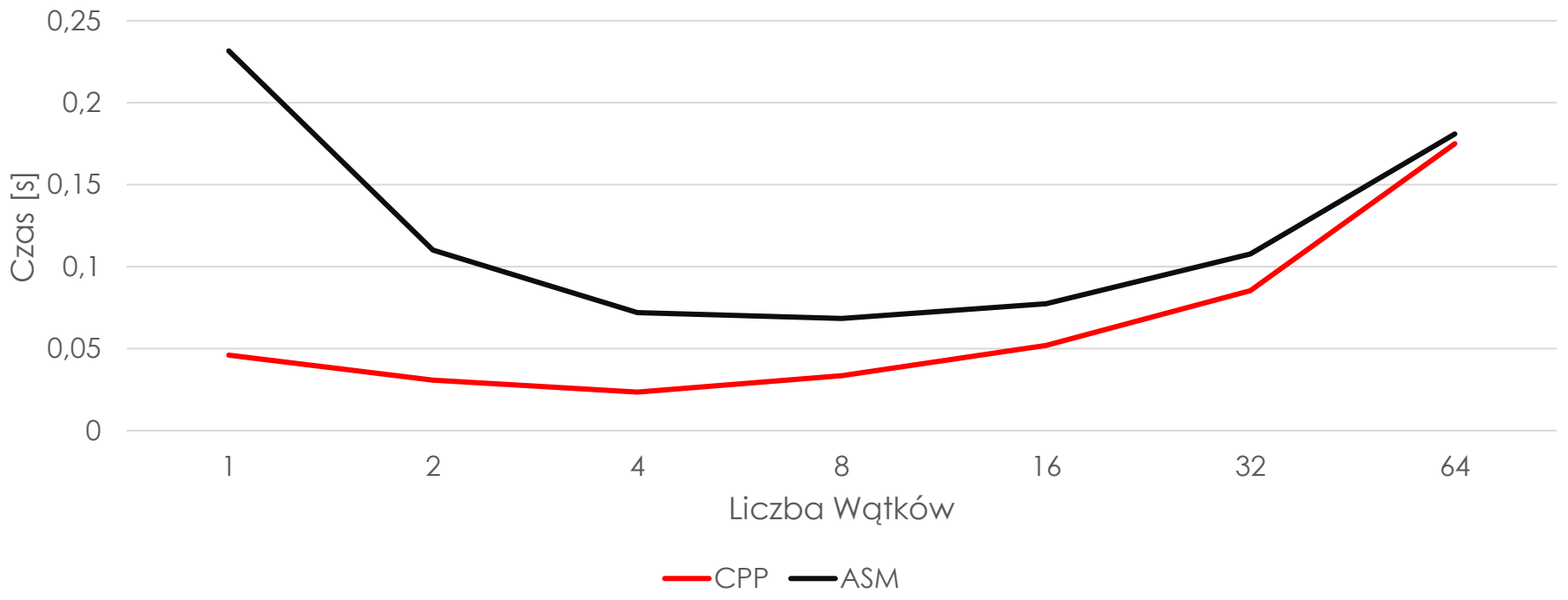
Porównanie czasów – 256x256



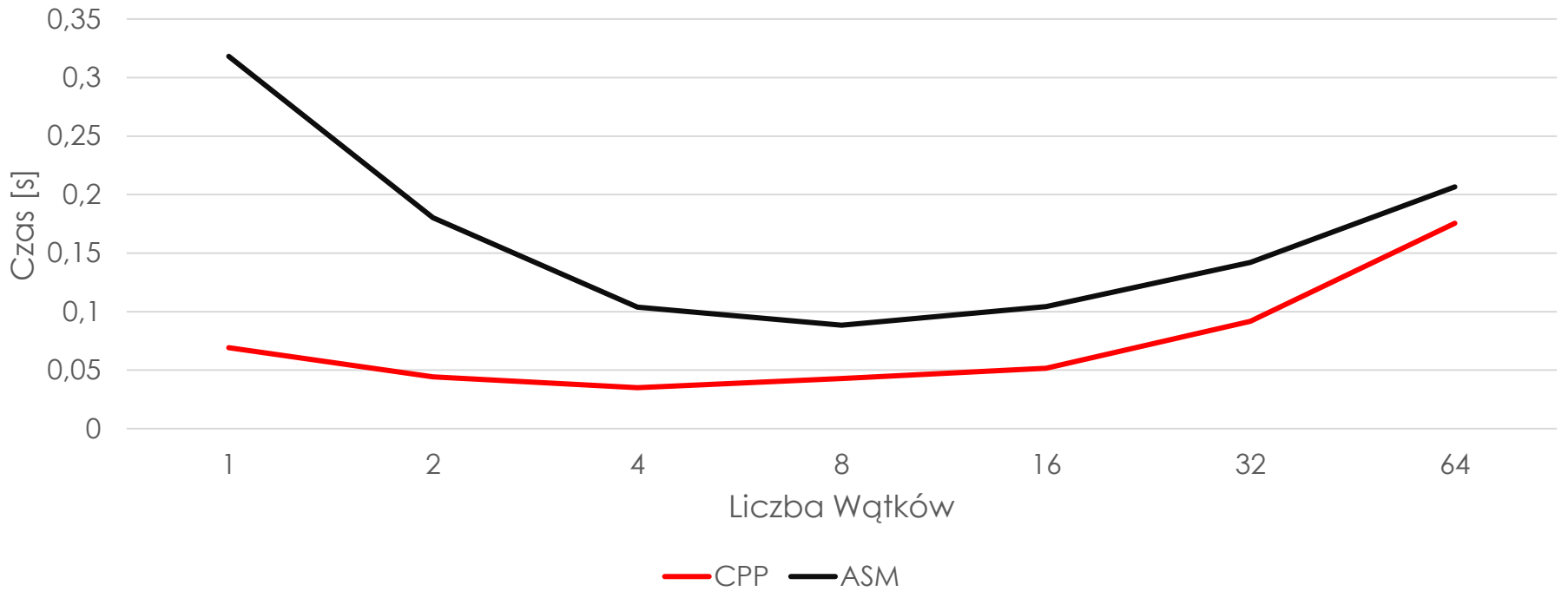
Porównanie czasów – 640x426



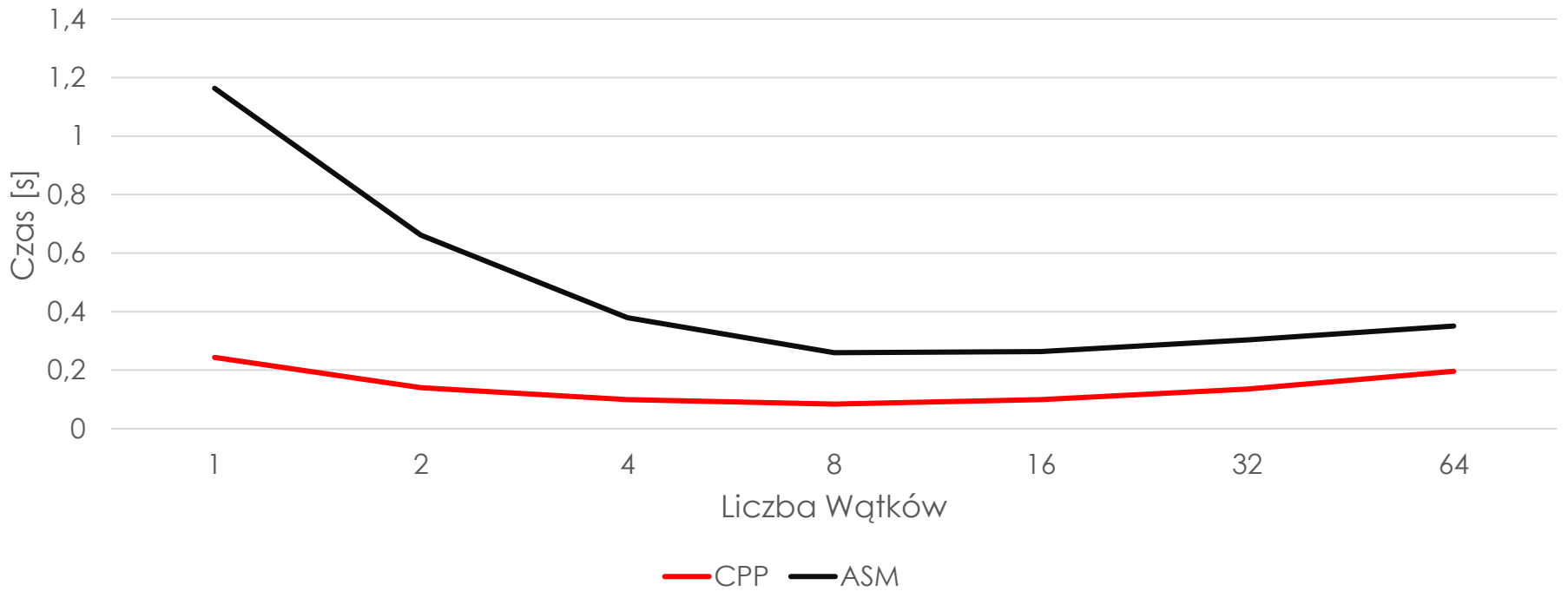
Porównanie czasów – 1280x1024



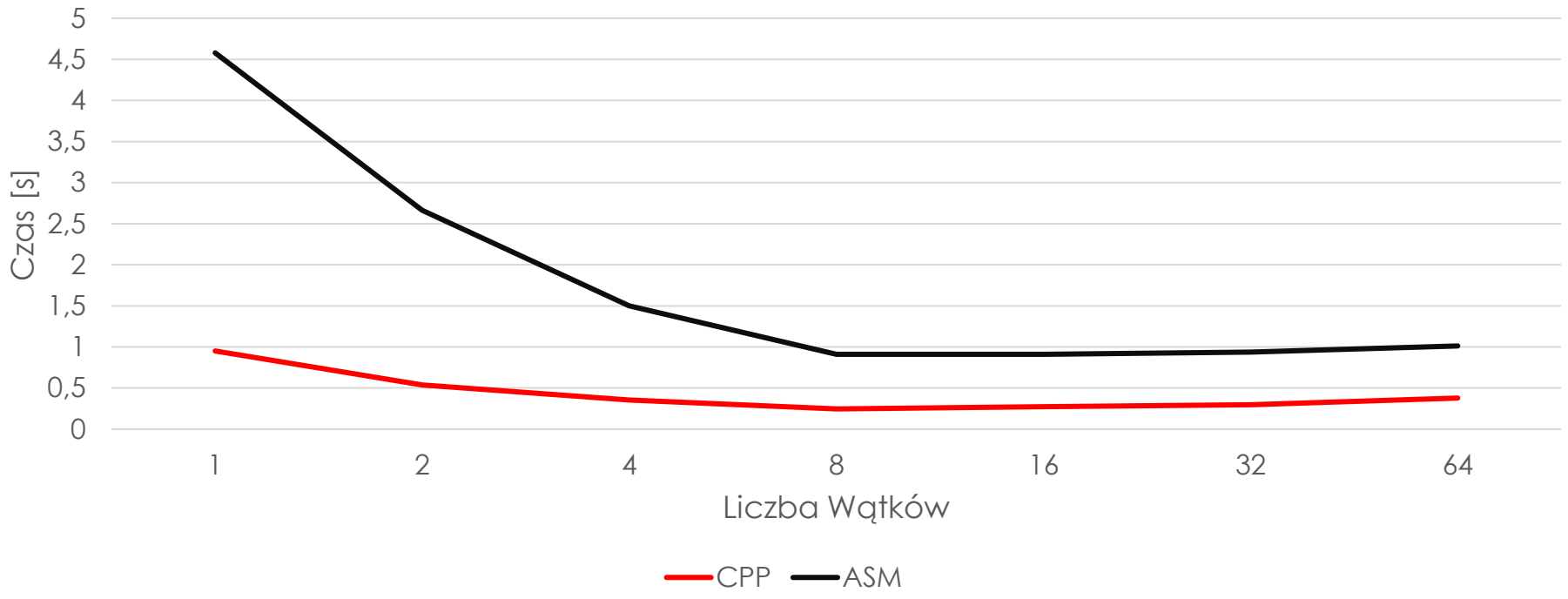
Porównanie czasów – 1920x1080



Porównanie czasów – 3840x2160



Porównanie czasów – 7680x4320



Wnioski

- Funkcja asemblerowa jest szybsza tylko dla małych obrazów i dużej ilości wątków
- Funkcje przeważnie działają najoptymalniej dla ośmiu wątków, zwłaszcza ta funkcja napisana w asemblerze
- Należy rozważnie wybierać ilość wątków za pomocą których filtrujemy obraz gdyż nie zawsze większa ilość wątków jest najefektywniejsza
- Największa różnica w czasach filtracji występuję na jednym wątku na korzyść CPP

Źródła

- https://pl.wikipedia.org/wiki/Rozmycie_gaussowskie
- http://www.algorytm.org/przetwarzanie-obrazow/filtrowanie-obrazow.html?fbclid=IwAR2koMcVECPQdnLRDIq5rRvK_X5MfmKP5qwQNT1xnopAEVsLGPa8dSlGeKI

Dziękuję za uwagę!

Autor: Kamil Niedziela