Silesian University
of Technology
Faculty of Automatic Control, Electronics and Computer Science
Department of Graphics, Computer Vision and Digital Systems

Rok akademicki	Rodzaj studiów*:	Przedmiot:	Grupa	Sekcja
2021/2022	**SSI**	**JA proj.**	**5**	**2**
Termin: (dzień, godzina)	**25.01.2022, 15:00**	**Prowadzący:**	**AO**	

Imię: Kamil
Nazwisko: Niedziela
Email: kaminie318@student.polsl.pl

Raport końcowy

Rozmycie Gaussowskie

1. Opis programu i założenia

Głównym celem stworzonego programu była implementacja algorytmu przetwarzającego obraz filtrem Gaussa. Algorytm przetwarza każdą składową obrazu RGB(red, green oraz blue) przez odpowiednią maskę. W przypadku tej implementacji została wykorzystana maska 3x3 następującej postaci:

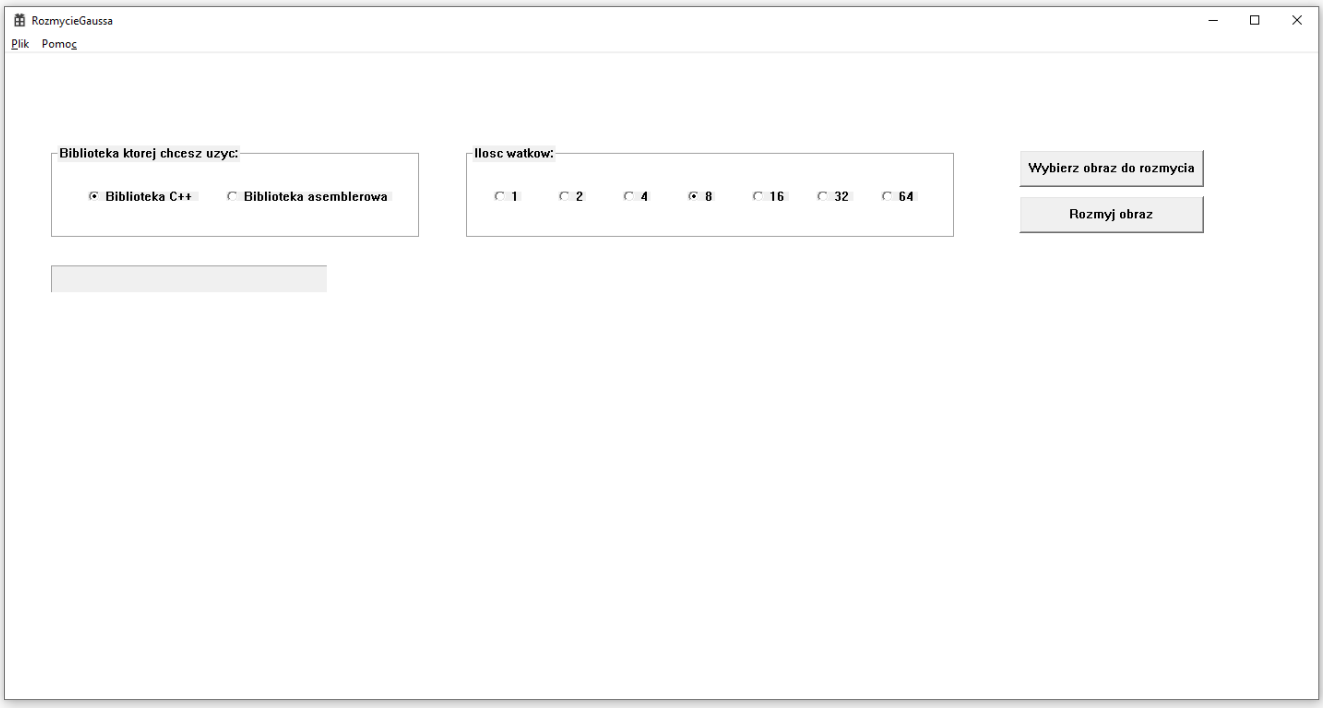
1	2	1
2	4	2
1	2	1

Suma ważona każdej składowej obrazu jest dzielona przez sumę wag maski. Przetworzone składowe zapisywane są w tablicy wyjściowej, wynikiem jest przetworzony obraz. Podczas implementacji należy uwzględnić momenty gdy maska wychodzi poza zakres obrazu (np. krawędzie).

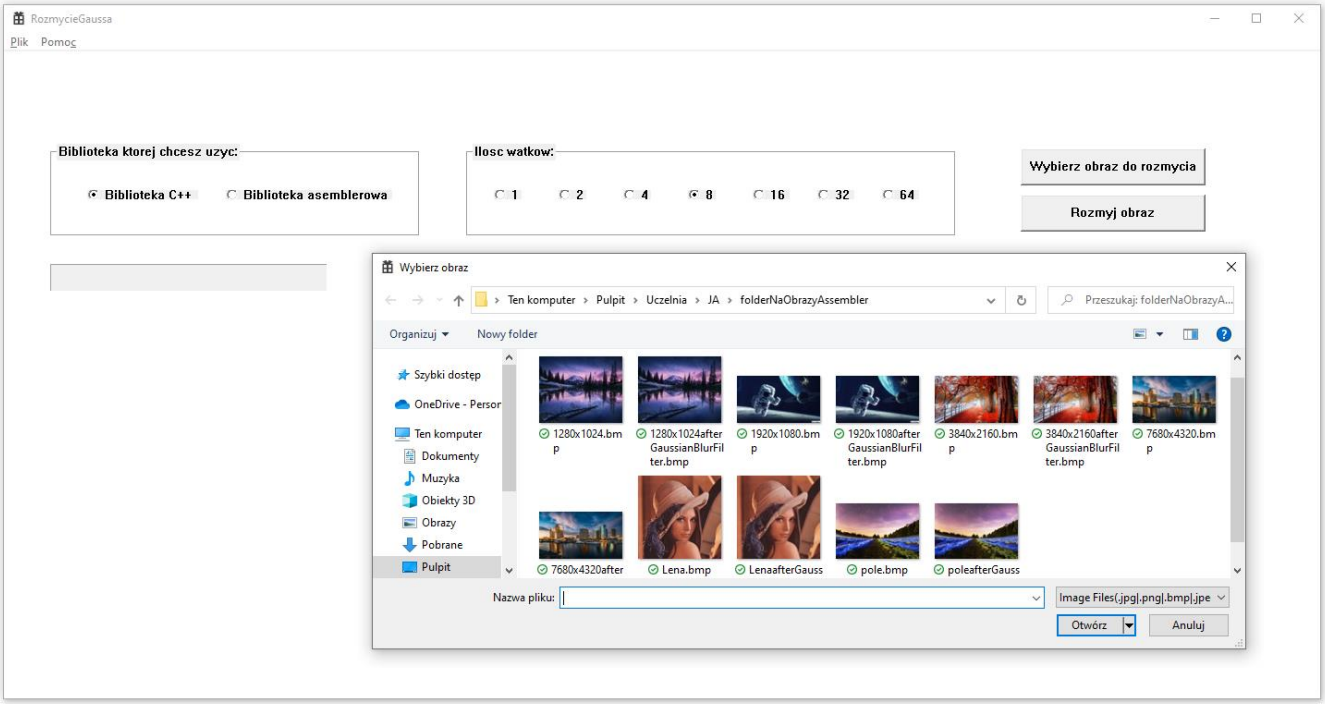
Użytkownik może wybrać ilość wątków za pomocą której filtracja ma zostać przeprowadzona(1, 2, 4, 8, 16, 32, lub 64 wątki). Użytkownik wybiera z której implementacji algorytmu chce skorzystać, tej napisanej w C++ czy tej napisanej w asemblerze. Osoba korzystająca z programu musi również wybrać którą bitmapę chce przetworzyć, umożliwia to przycisk otwierający menadżer katalogów za pomocą którego można wybrać odpowiednią bitmapę. Po przetworzeniu obrazu w tym samym folderze tworzy się przetworzona bitmapa z przyrostkiem „NAZWAOBRAZUafterGaussianBlurFilter.bmp”

2. Graficzny interfejs użytkownika

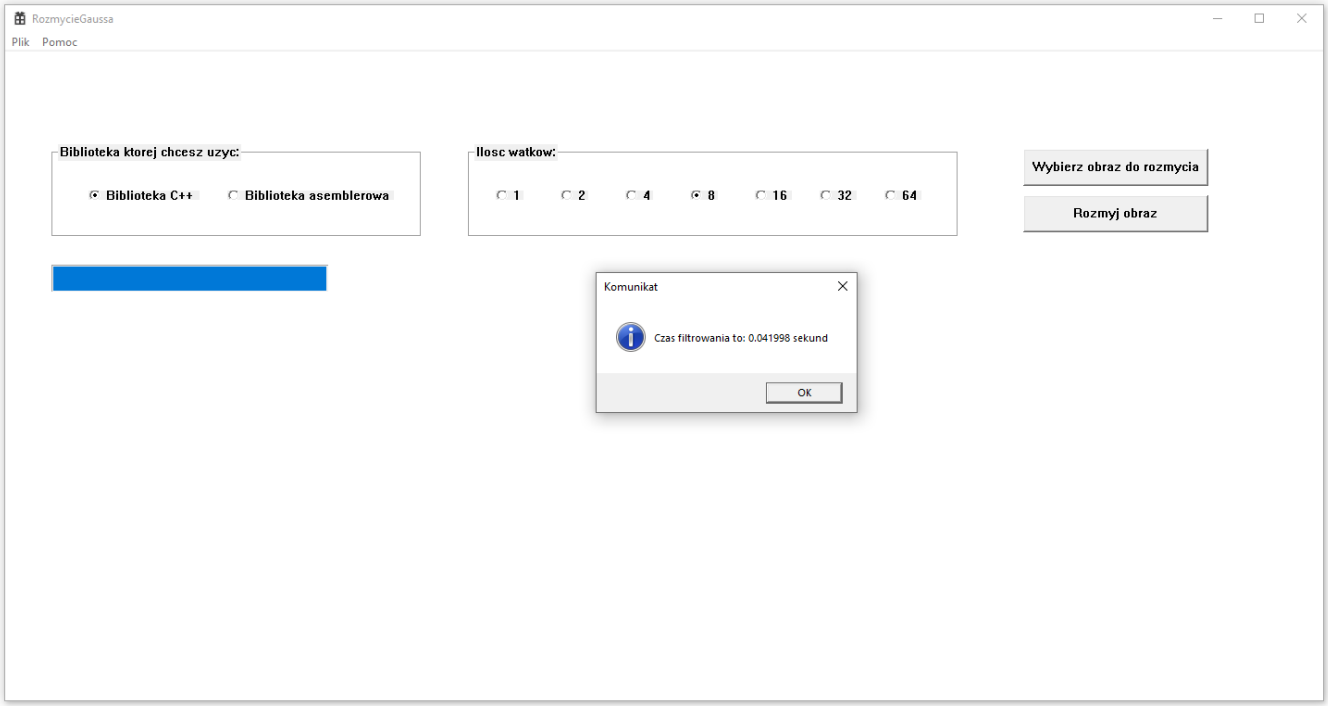
Ekran startowy:



Wybór bitmapy:



Po filtracji:



Przetworzony obraz znajduję się w tym samym katalogu gdzie znajduje się bitmapa. Widoczny pasek postępu pokazuje progres rozmycia.

3. Parametry wejściowe programu

Wartości które przekazywane są do funkcji rozmywającej to wskaźnik na tablicę wejściową, wskaźnik na tablicę wyjściową, szerokość obrazu, wysokość obrazu od której filtrowanie ma się rozpocząć, wysokość obrazu gdzie filtrowanie ma się skończyć. Parametrami wejściowymi w programie są pliki w formacie .bmp. Po wybraniu bitmapy program wczytuje piksele do tablicy wejściowej. Następnie program przetwarza bitmapę i rozmyty już obraz zapisuje do tablicy wyjściowej.

4. Fragment kodu w języku asemblera

```
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
```

Podany fragment sprawdza czy maska która filtruję znajduję się w obszarze obrazka. Jeżeli przetwarzany piksel znajduję się np. na krawędzi lub w którymś z rogów obrazu niektóre wartości maski podczas filtracji są ignorowane.

5. Testowanie programu

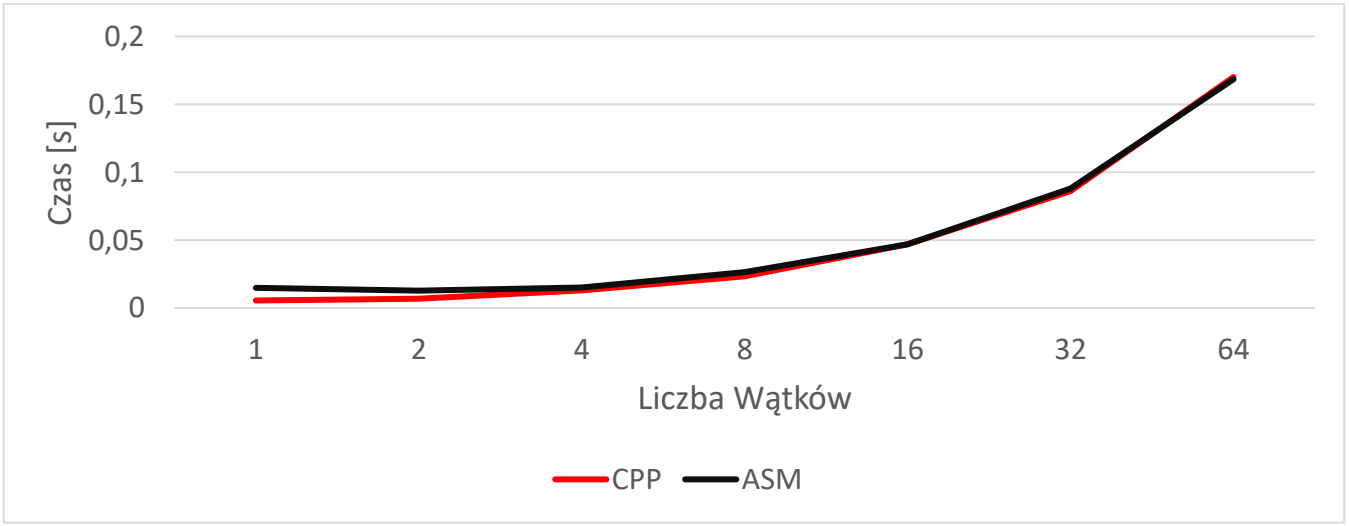
Program został przetestowany dla bitmap o różnej rozdzielczości takich jak: 256x256, 640x426, 1280x1024, 1920x1080, 3840x2160, 7680x4320. Program został zabezpieczony przed innymi danymi wejściowymi niż bitmapa.

6. Pomiary czasu

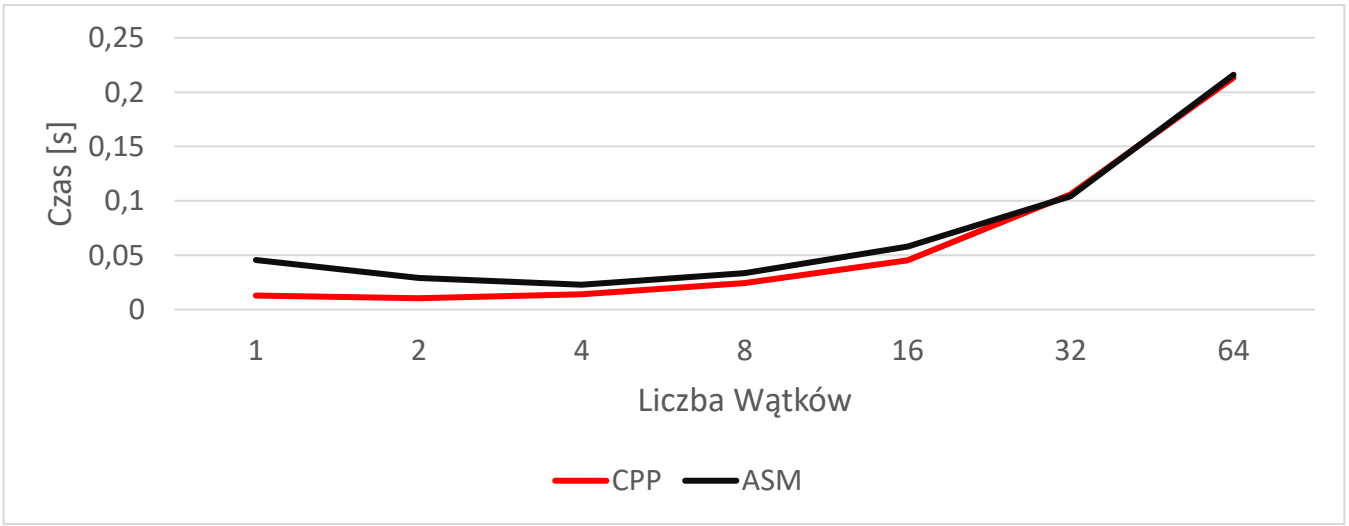
Pomiary wykonano dla małych, średnich i dużych bitmap dla każdego wątku. Do stworzenia wykresów wykorzystano średnią z trzech pomiarów.

Małe bitmapy:

- 256x256

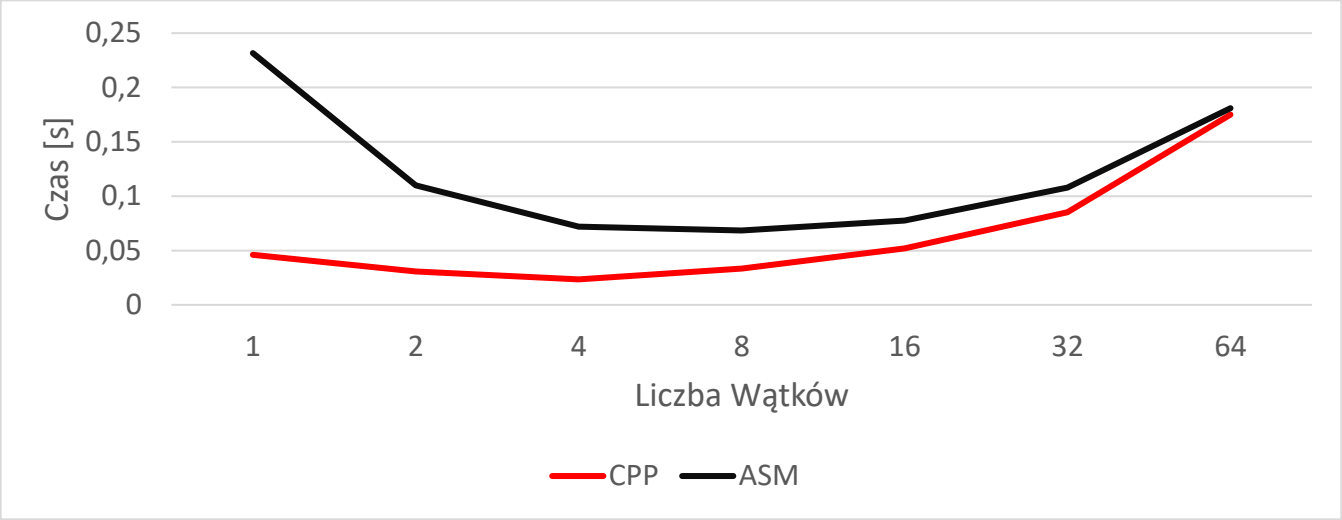


- 640x426

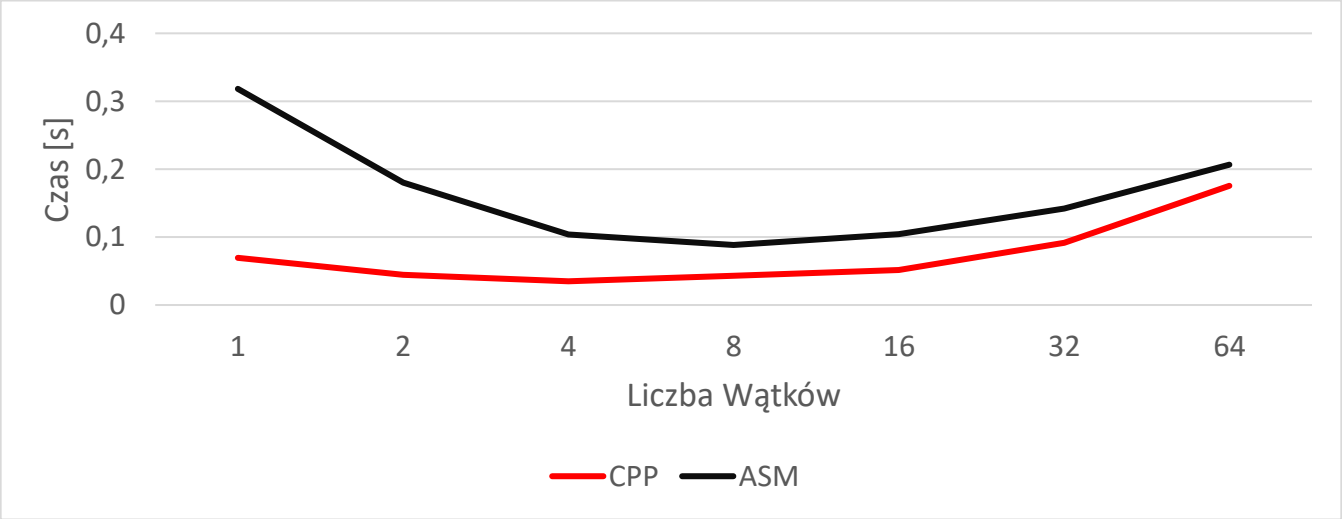


Średnie bitmapy:

- 1280x1024

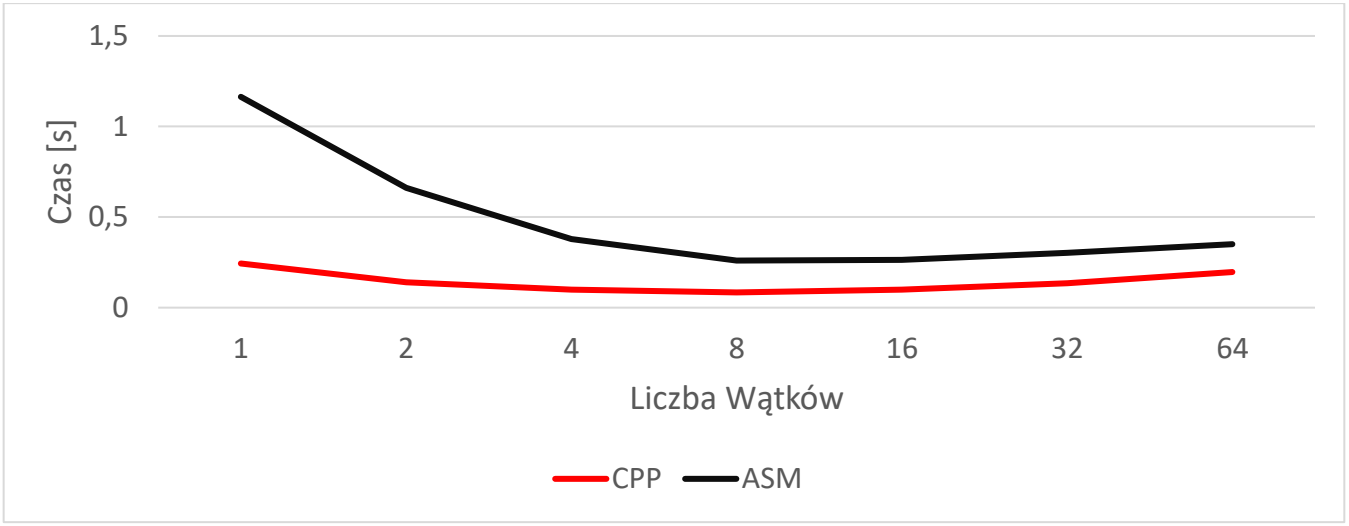


- 1920x1080

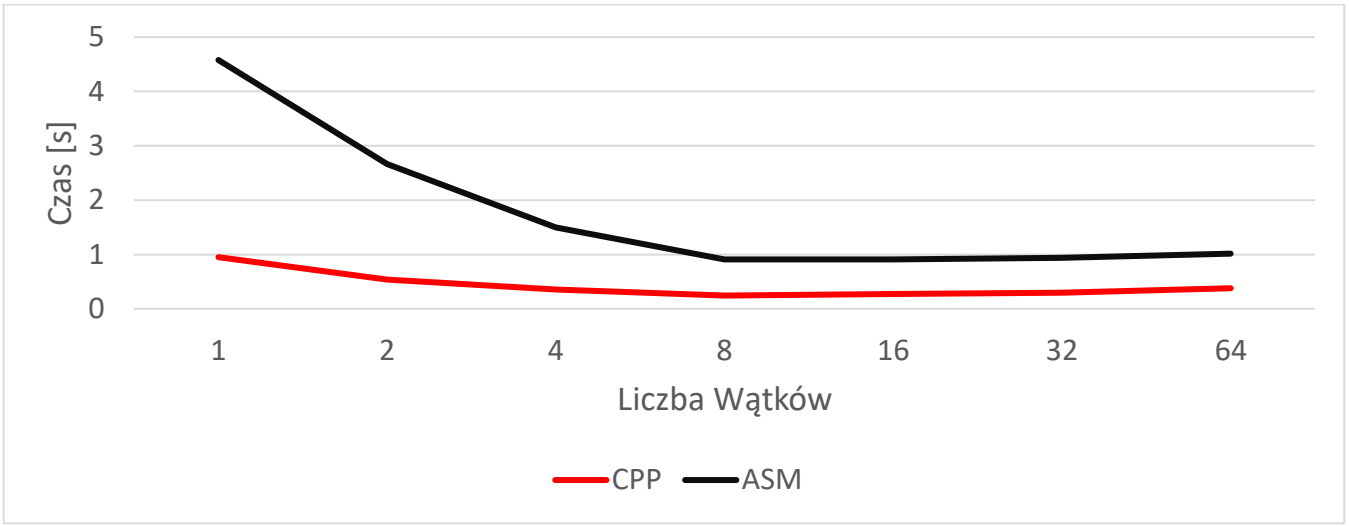


Duże bitmapy:

- 3840x2160



- 7680x4320



7. Wnioski

Funkcja napisana w C++ została napisana lepiej od tej asemblerowej. Rozmycie każdej rozdzielczości na każdym wątku wypada lepiej z wykorzystaniem funkcji napisanej w języku wysokiego poziomu. Wyjątkiem od reguły jest przetwarzanie małych obrazów dla wielu wątków, tam możemy dostrzec delikatne zwycięstwo asemblera. Największą dysproporcję czasową pomiędzy funkcjami można dostrzec na jednym wątku. W ogólnym rozrachunku mimo wszystko lepiej korzystać z funkcji napisanej w C++. Warto również wspomnieć że wybrana największa ilość wątków wcale nie oznacza najszybszego czasu filtracji. Taką zależność można dostrzec na powyższych wykresach. Należy znaleźć złoty środek pomiędzy wielkością obrazu a wybieraną ilością wątków. Dostępna ilość procesów logicznych(wątków) w procesorze ma również ogromny wpływ na osiągane rezultaty. Programowanie w asemblerze sprawia że mamy pełną kontrolę nad rozkazami wydawanymi przez procesor. Pisanie kodu w asemblerze z pewnością może skrócić czas wykonania programu, jednak wymaga znacznie większego skupienia niż podczas pisania w języku wysokiego poziomu jak np. C++, JAVA. Mimo wszystko uważam zadanie za ciekawe doświadczenie, wykonanie projektu znacznie poszerzyło moją wiedzę dotyczącą programowania w języku asemblera.