|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| logowydzialu | 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:**  **Nazwisko:**  **Email:** | **Kamil**  **Niedziela**  **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:

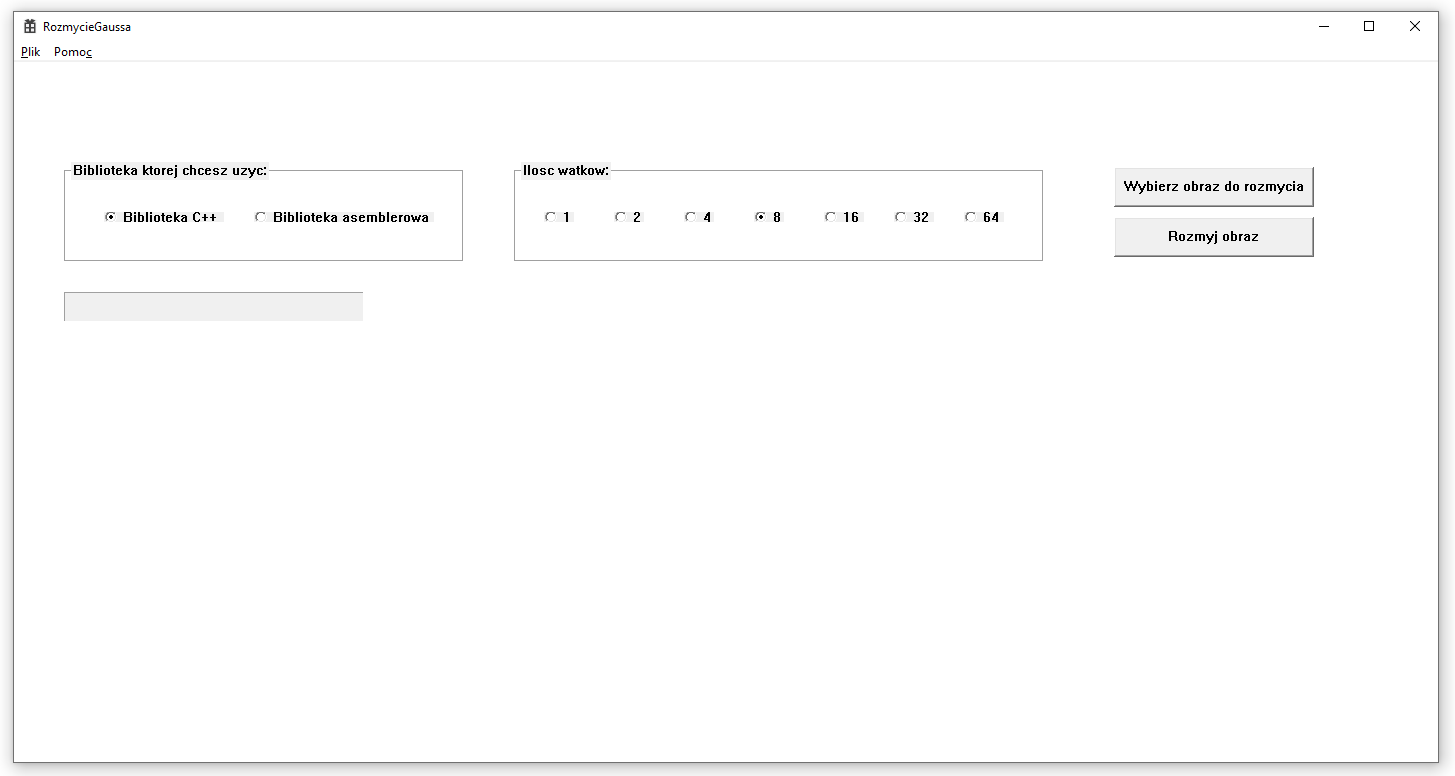


Suma ważona każdej składowej obrazu jest dzielona przez sumę wag maski. Przetworzone składowe zapisywane są w tablicy wyjściowej, wynikiem jest przetworzony obraz. Podczas implementacji należy uwzględnić momenty gdy maska wychodzi poza zakres obrazu (np. krawędzie).

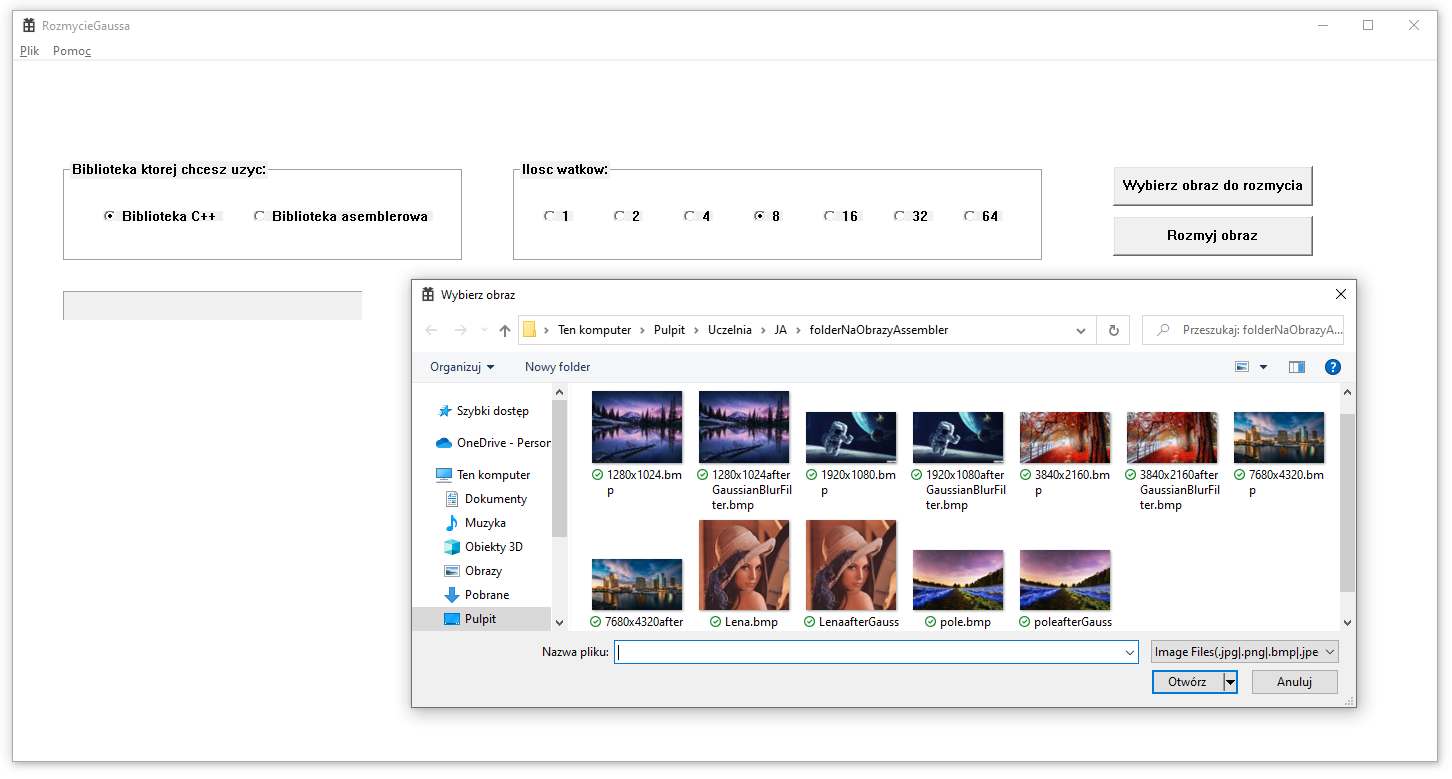
Użytkownik może wybrać ilość wątków za pomocą której filtracja ma zostać przeprowadzona(1, 2, 4, 8, 16, 32, lub 64 wątki). Użytkownik wybiera z której implementacji algorytmu chce skorzystać, tej napisanej w C++ czy tej napisanej w asemblerze. Osoba korzystająca z programu musi również wybrać którą bitmapę 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**

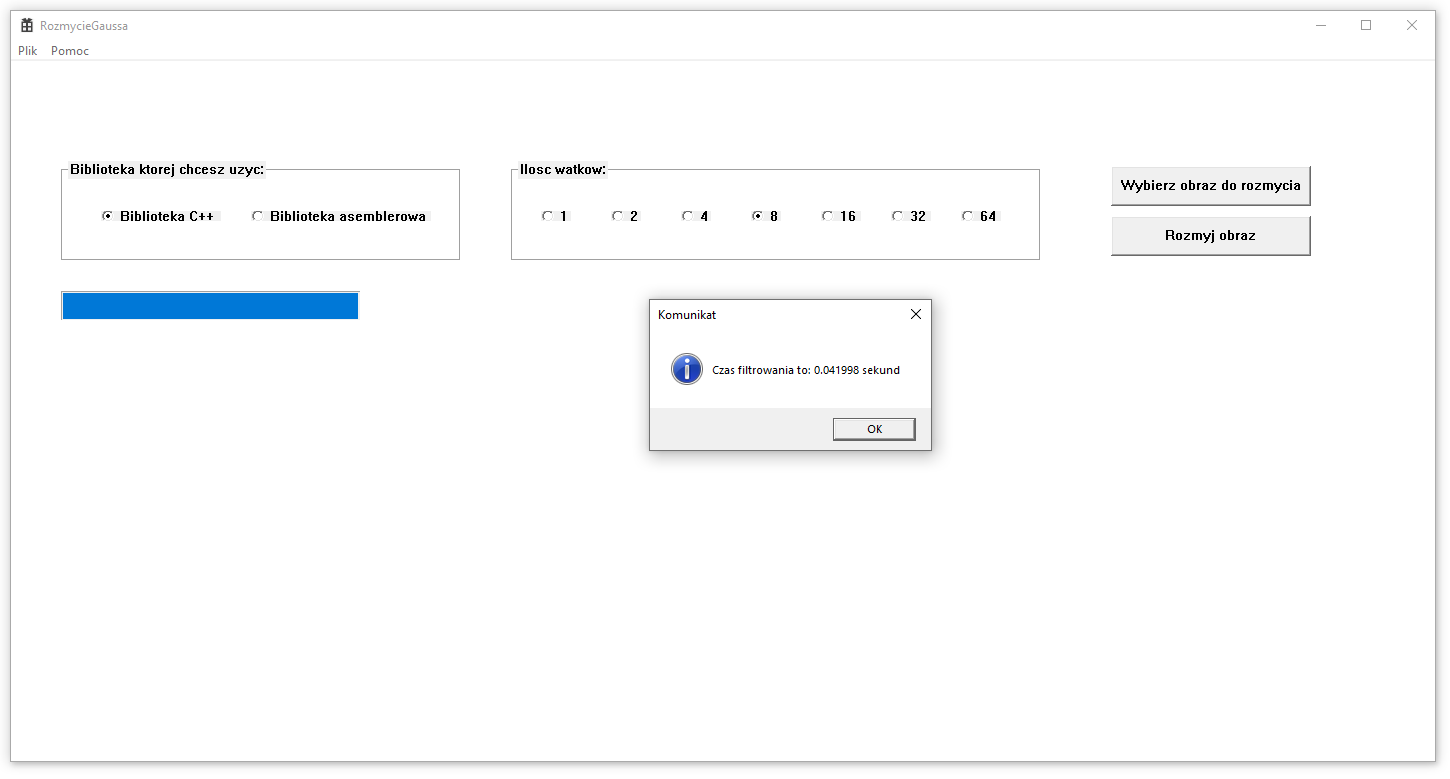
Ekran startowy:



Wybór bitmapy:



Po filtracji:

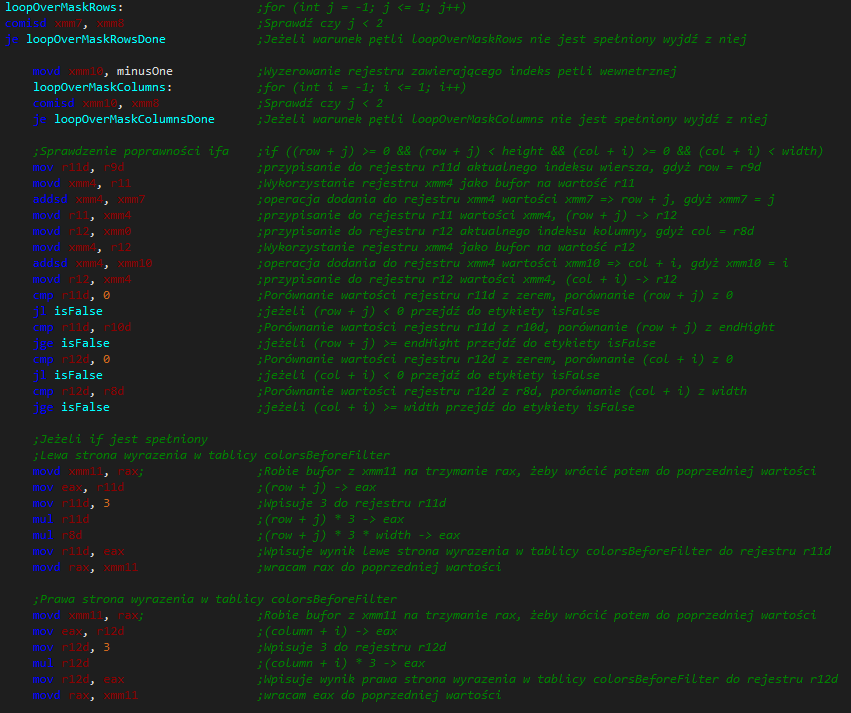


Przetworzony obraz znajduję się w tym samym katalogu gdzie znajduje się bitmapa. Widoczny pasek postępu pokazuje progres rozmycia.

**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**



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.