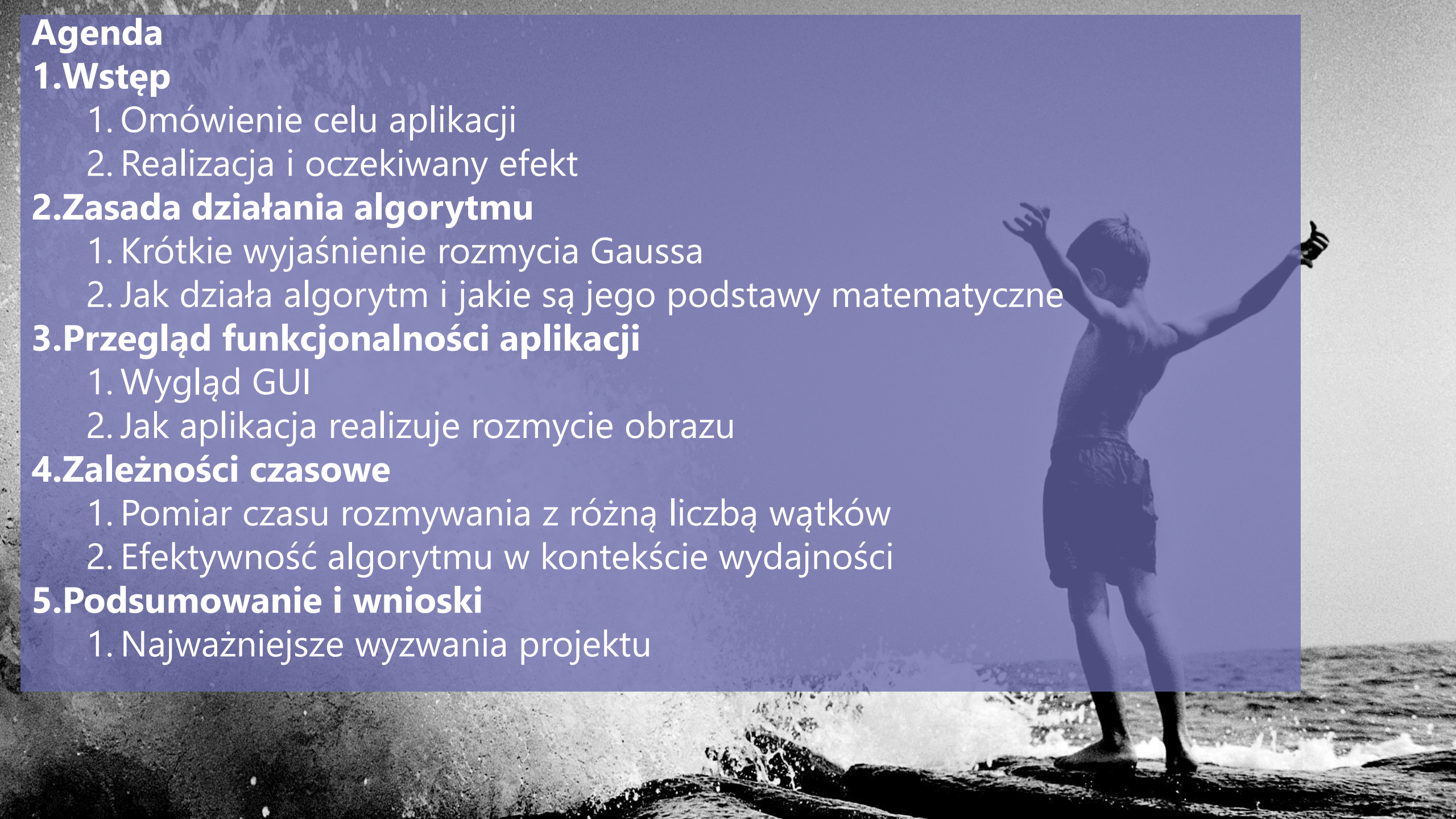
ROZMYCIE GAUSSA



Agenda

1.Wstęp

1. Omówienie celu aplikacji
2. Realizacja i oczekiwany efekt

2.Zasada działania algorytmu

1. Krótkie wyjaśnienie rozmycia Gaussa
2. Jak działa algorytm i jakie są jego podstawy matematyczne

3.Przegląd funkcjonalności aplikacji

1. Wygląd GUI
2. Jak aplikacja realizuje rozmycie obrazu

4.Zależności czasowe

1. Pomiar czasu rozmywania z różną liczbą wątków
2. Efektywność algorytmu w kontekście wydajności

5.Podsumowanie i wnioski

1. Najważniejsze wyzwania projektu

1. Cel aplikacji:

- Automatyzacja rozmycia obrazu: Tworzymy aplikację, która wykonuje rozmycie Gaussa na obrazach w sposób efektywny, wykorzystując algorytm wielowątkowy.

- **Zoptymalizowana wydajność**: Używamy wielowątkowości oraz instrukcji wektorowych, aby przyspieszyć proces rozmywania, szczególnie przy dużych obrazach.

2. Oczekiwany efekt:

- Szybsze przetwarzanie: Rozmycie obrazu za pomocą algorytmu Gaussa, który przetwarza obrazy na poziomie pojedynczych pikseli, znacznie szybciej niż tradycyjne metody.

- Wielowątkowość: Wydajność aplikacji jest dostosowywana do liczby rdzeni procesora, z automatycznym wykrywaniem liczby dostępnych rdzeni i ustawieniem odpowiedniej liczby wątków.

Opis algorytmu rozmycia Gaussa

Rozmycie Gaussa jest techniką stosowaną w grafice komputerowej, która wygładza obraz, usuwając szczegóły i szumy, na podstawie funkcji Gaussa.

Matematyka w algorytmie: Algorytm przetwarza każdy piksel w obrazie, uśredniając wartości pikseli w jego sąsiedztwie (wokół okna filtrującego o zadanym rozmiarze).

- Przetwarzanie pikseli:

Dla każdego piksela obrazu, obliczane są średnie wartości dla kanałów **R (czerwony)**, **G (zielony)**, i **B (niebieski)**, bazując na wartościach pikseli w obrębie określonego bloku (okna).

- **Obliczanie średniej**: Dla każdego piksela w obrębie bloku, algorytm sumuje wartości poszczególnych kanałów, a następnie dzieli je przez liczbę rozważanych pikseli, uzyskując średnią dla każdego kanału.

Instrukcje wektorowe (SIMD):

- SIMD (Single Instruction, Multiple Data) pozwala na równoległe przetwarzanie danych. Dzięki instrukcjom takim jak _mm_add_pd, _mm_cvtepi32_pd, aplikacja przetwarza wiele danych jednocześnie, przyspieszając obliczenia.

- Użycie SIMD umożliwia równoczesne przetwarzanie 4 wartości w jednym cyklu, co znacznie zwiększa wydajność algorytmu, szczególnie przy dużych obrazach.

Implementacja w C++

Przetwarzanie w obrębie bloków:
Algorytm działa na obrazach w blokach o określonym rozmiarze. Dla każdego piksela obliczana jest średnia wartość kanałów (R, G, B) na podstawie sąsiednich pikseli w obrębie bloku (np. 3x3).
Optymalizacja za pomocą instrukcji wektorowych:
Wykorzystanie SIMD (Single Instruction, Multiple Data) pozwala na równoległe przetwarzanie wielu pikseli.
Zamiast przetwarzać każdy piksel osobno, algorytm wczytuje i przetwarza grupy pikseli w jednym cyklu procesora (wektoryzacja).
Przykład działania:
Dla każdego piksela obliczane są jego sąsiedzi w obrębie rozmycia (w zależności od rozmiaru bloku). Następnie rozkładamy dane na kanały RGB i sumujemy ich wartości co pozwala na obliczenie średnich dla każdego kanału (R, G, B).
Optymalizacja wątków:
Algorytm rozmywa obraz równolegle w kilku wątkach, co pozwala na wykorzystanie mocy obliczeniowej wielu rdzeni procesora. Wykorzystanie 1-64 wątków w zależności od dostępnych procesorów logicznych pozwala na szybkie przetwarzanie dużych obrazów.

Implementacja w ASM

Przygotowanie parametrów:
- Wejście: Bufor pikseli obrazu (InBuffer), wysokość, szerokość obrazu oraz indeksy początkowy i końcowy zakresu przetwarzania.
- Wyjście: Bufor wynikowy (OutBuffer) z przetworzonymi pikselami.
- Parametry rozmycia: Rozmiar okna rozmycia (blockSize).

Podział przetwarzania na wiersze:
- Obliczane są współrzędne startowe (startY) oraz końcowe (endY) na podstawie indeksów zakresu.
- Algorytm iteruje przez piksele w obrębie podanego zakresu wierszy.

Rozmycie Gaussa:
- Obliczanie średniej: Piksele w obrębie okna (blockSize x blockSize) są sumowane dla każdego kanału (R, G, B) przy użyciu rejestrów SIMD (xmm0, xmm1, xmm2).
- Średnia jest obliczana przez podzielenie sumy przez liczbę pikseli w oknie.
- Obsługa brzegów: Piksele na brzegach obrazu, które nie mają pełnego sąsiedztwa, są kopiowane bez zmian.

Optymalizacja z użyciem SIMD:
- Zastosowanie rejestrów SIMD pozwala na równoczesne przetwarzanie danych w kanałach kolorów.
- Instrukcje SIMD (addss, divss) przyspieszają obliczenia, szczególnie dla dużych obrazów.

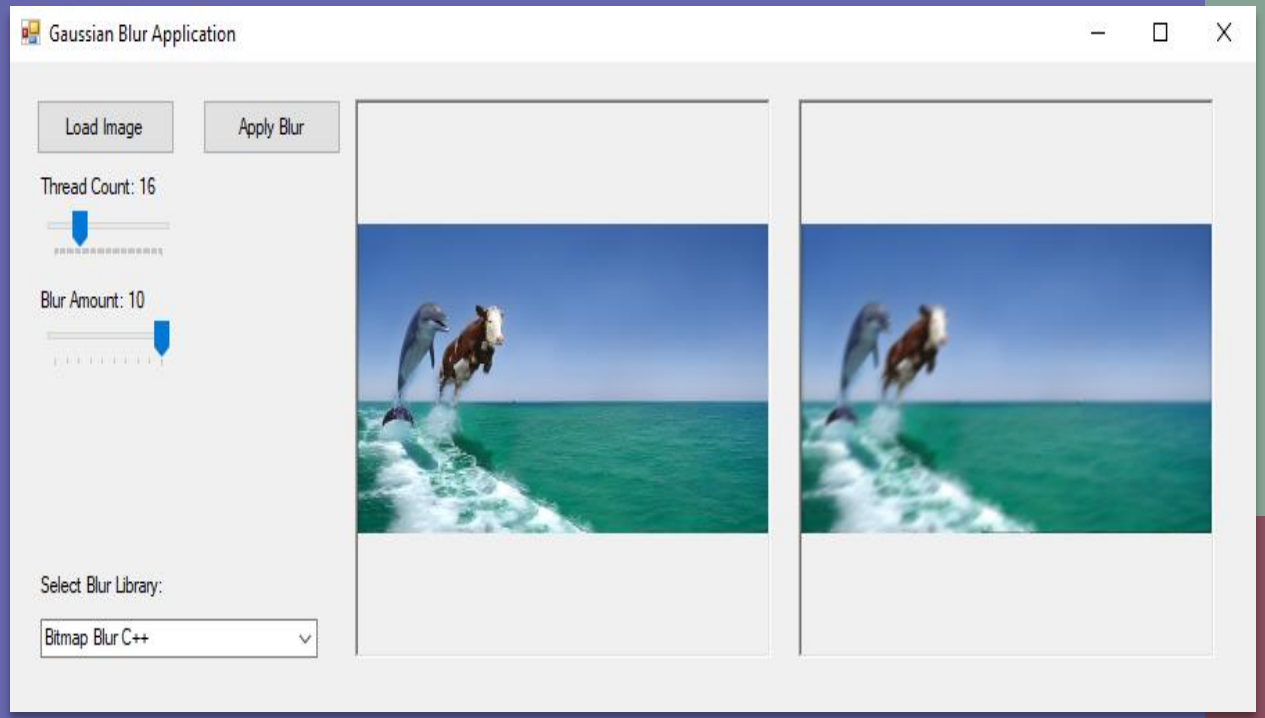
Kodowanie wyniku:
- Wyniki dla kanałów R, G, B są zaokrąglane i zapisywane do bufora wyjściowego.

INTERFEJS UŻYTKOWNIKA

Omówienie funkcjonalności:

- **Load Image**: Przycisk pozwalający użytkownikowi na załadowanie obrazu w formacie JPG lub PNG do przetworzenia.
- **Apply Blur**: Po załadowaniu obrazu i wybraniu reszty ustawień użytkownik może kliknąć przycisk "Rozmyj", aby uruchomić proces.
- **Thread Count**: Użytkownik może wybrać liczbę wątków (od 1 do 64), co automatycznie dostosowuje aplikację do liczby rdzeni procesora.
- **Blur Amount**: Użytkownik może dostosować stopień rozmycia.
- **Select Blur Library**: Aplikacja umożliwia wybór między biblioteką C++ a ASM, dając użytkownikowi możliwość wyboru rozwiązania.

Po prawej stronie możemy zobaczyć gotowy interfejs użytkownika, w którym są zaimplementowane funkcjonalności. Obraz przed rozmyciem jest wyświetlany w małym okienku po lewej stronie, a przetworzony obraz jest widoczny w okienku po prawej stronie.



Omówienie zależności czasowych

Wyniki pokazują, że czas przetwarzania znacząco maleje wraz ze wzrostem liczby wątków w obu przypadkach, co świadczy o efektywnym wykorzystaniu wielowątkowości.

Dla biblioteki C++ czas przetwarzania zaczyna się od 760 ms przy jednym wątku i systematycznie spada, osiągając 113 ms przy 64 wątkach. Największy spadek czasu widoczny jest w zakresie od 1 do 16 wątków, natomiast powyżej 16 wątków dalsze optymalizacje są marginalne, co może wynikać z ograniczeń sprzętowych, takich jak liczba fizycznych rdzeni procesora lub narzut związany z zarządzaniem wielowątkowością.

Biblioteka ASM wykazuje podobny trend, jednak czasy przetwarzania są znacząco niższe we wszystkich testowanych konfiguracjach. Przy jednym wątku czas wynosi 455 ms, a przy 64 wątkach – jedynie 58 ms. Różnica ta wynika z charakteru assemblera, który pozwala na bardziej zoptymalizowane i bezpośrednie wykorzystanie zasobów procesora.

Wnioski:
Porównując obie biblioteki, można zauważyć, że ASM jest wydajniejszy już od najniższej liczby wątków. Dla jednego wątku jest około 1,67 raza szybszy (455 ms w porównaniu do 760 ms), a przy 64 wątkach różnica wciąż się utrzymuje, choć zmniejsza się do około 1,95 raza. Pokazuje to, że biblioteka assemblerowa jest lepszym wyborem w sytuacjach, gdy kluczowa jest maksymalna wydajność.

Biblioteka C++		Biblioteka ASM	
liczba wątków	czas (ms)	liczba wątków	czas (ms)
1	760	1	455
2	391	2	262
4	285	4	136
8	186	8	88
16	123	16	67
32	119	32	60
64	113	64	58

Podsumowanie

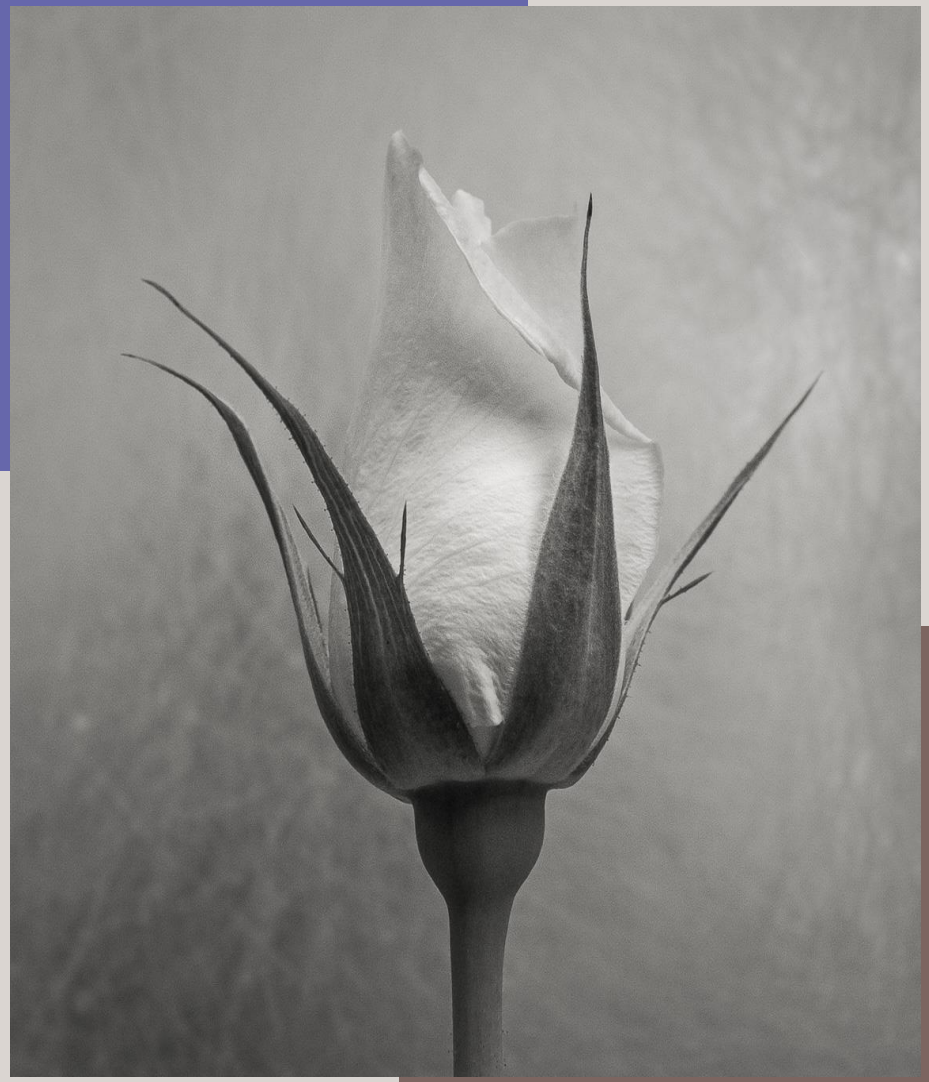
Zrealizowane funkcjonalności:

- Wczytywanie obrazów z plików.
- Możliwość wyboru liczby wątków (od 1 do 64),
- Zastosowanie dwóch bibliotek rozmycia: C++ i ASM
- Wykorzystanie instrukcji wektorowych, co przyspiesza operacje.

Wnioski:

- Wydajność: Aplikacja skutecznie wykorzystuje wiele rdzeni procesora, co pozwala na przyspieszenie operacji rozmycia, szczególnie w przypadku obrazów o większych rozdzielczościach.

- Zastosowanie w praktyce: Dzięki możliwości dostosowania liczby wątków oraz wyboru biblioteki, aplikacja jest elastyczna i dostosowuje się do różnych środowisk i potrzeb użytkownika.



DZIĘKUJĘ ZA UWAGĘ

Jakub Różycki
Jr305142@student.polsl.pl
KIERUNEK INFORMATYKA KATOWICE
Rok Akademicki 2024/2025
Grupa dziekańska 1
Sekcja 2
Semestr 5