

1. Interfejs użytkownika

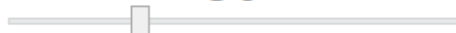
Add noise to the image

Noise type:

- ☒ Random noise
- ☐ White noise
- ☐ Color noise

Noise Power:

30

A horizontal slider bar with a small square handle positioned at approximately one-third of the way from the left.

Generating library:

- ☒ Assembly
- ☐ C++

Number of threads:

8

A horizontal slider bar with a small square handle positioned at approximately one-third of the way from the left.

Measure time

SaveSelect

Add chosen noise

Last generation time:
00s 000ms

Z menu można wybrać jeden z trzech rodzajów szumu za pomocą radio button, który chce się dodać do bitmapy. Wybór biblioteki działa w ten sam sposób. Domyślnie ustawione są wartości przedstawione na grafice, tak żeby niemożliwym było niezaznaczenie żadnych opcji. Kolejne parametry obsługiwane są przez slidery. Dla Noise Power jest to slider z zakresu 1-100 i określa on natężenie i siłę zaszumienia w przypadku wybrania szumu białego i kolorowego. Number of threads określa ilość wątków, na których zostanie wykonana operacja. W tym przypadku zakres jest z zakresu 1-64, a domyślna wartość określana jest na podstawie liczbie procesorów logicznych komputera, na którym pracuje aplikacja. Za pomocą przycisku Measure Time można zmierzyć średni czas wywołań programu dla obu bibliotek dla wątków 1, 2, 4, 8, 16, 32, 64 i zobaczyć porównanie. Dla przycisku Save podpiętą funkcją jest zapisanie przetworzonej bitmapy do katalogu outputs, znajdującego się w katalogu projektu. Przycisk Select pozwala na wybranie bitmapy do przetworzenia. Add chosen noise dodaje szum zgodnie z wybranymi parametrami. No dole można zaobserwować ramkę z wynikiem czasowym ostatniego wywołania. Aplikacja jest zabezpieczona przed próbą dodania szumu bez wcześniejszego wybrania bitmapy.

2. Opis realizowanych szumów

Aplikacja realizuje trzy rodzaje szumów:

- a. **Random** – szum losowy, który działa w sposób, że losowe piksele w losowej ilości ustawiane są na losowe kolory
- b. **White** – szum biały, który polega na dodaniu losowych odchyłeń wartości RGB pikseli
- c. **Color** – szum kolorowy, do którego generowany jest losowy kolor i w losowych odchyleniach jego wartości dodawany jest on do bitmapy

3. Fragmenty kodu

```
.code
whiteNoiseAsm PROC
    movups xmm0, [rcx]           ; wczytanie u1

    ;mnozenie xmm0 * (-2)
    movups xmm1, xmm0
    movss xmm2, negative_two
    shufps xmm2, xmm2, 0
    mulps xmm1, xmm2
    sqrtps xmm1, xmm1

    movups xmm0, [rcx + 16]      ; wczytanie u2
    mulps xmm1, xmm0            ; z = u1*u2

    cvtsi2ss xmm0, r8           ; noisePower
    mulps xmm1, xmm0            ; z*=noisePower

    movups xmm2, [rdx]          ; wczytanie r
    movups xmm3, [rdx + 16]     ; wczytanie g
    movups xmm4, [rdx + 32]     ; wczytanie b

    addps xmm2, xmm1             ; r = r + z
    addps xmm3, xmm1             ; g = g + z
    addps xmm4, xmm1             ; b = b + z
```

W tym fragmencie kodu assemblerowego wykonywana jest część operacji dodawania białego szumu. Poprzez wykorzystanie instrukcji wektorowych operacje wykonywane są szybciej i można pracować na wielu danych w jednym czasie.

```

movups xmm0, [maxes]
minps xmm2, xmm0
minps xmm3, xmm0
minps xmm4, xmm0

movups xmm0, [mins]
maxps xmm2, xmm0
maxps xmm3, xmm0
maxps xmm4, xmm0

```

Ten fragment kodu standaryzuje wektor wynikowy do zakresu liczbowego w zakresie 0-255, tak żeby wartości mieściły się w granicach kolorów RGB.

4. Pomiary czasowe

Dane małe(~7KB)

×

1 threads, generated in 151 ms using the assembly library.
 1 threads, generated in 240 ms using the C++ library.
 2 threads, generated in 183 ms using the assembly library.
 2 threads, generated in 209 ms using the C++ library.
 4 threads, generated in 191 ms using the assembly library.
 4 threads, generated in 198 ms using the C++ library.
 8 threads, generated in 209 ms using the assembly library.
 8 threads, generated in 224 ms using the C++ library.
 16 threads, generated in 195 ms using the assembly library.
 16 threads, generated in 239 ms using the C++ library.
 32 threads, generated in 203 ms using the assembly library.
 32 threads, generated in 218 ms using the C++ library.
 64 threads, generated in 187 ms using the assembly library.
 64 threads, generated in 238 ms using the C++ library.

OK

Dane średnie(~7000KB)

×

1 threads, generated in 497 ms using the assembly library.
 1 threads, generated in 640 ms using the C++ library.
 2 threads, generated in 465 ms using the assembly library.
 2 threads, generated in 697 ms using the C++ library.
 4 threads, generated in 537 ms using the assembly library.
 4 threads, generated in 708 ms using the C++ library.
 8 threads, generated in 532 ms using the assembly library.
 8 threads, generated in 720 ms using the C++ library.
 16 threads, generated in 523 ms using the assembly library.
 16 threads, generated in 610 ms using the C++ library.
 32 threads, generated in 536 ms using the assembly library.
 32 threads, generated in 722 ms using the C++ library.
 64 threads, generated in 508 ms using the assembly library.
 64 threads, generated in 647 ms using the C++ library.

OK

Dane duże (~43MB)



1 threads, generated in 1528 ms using the assembly library.
1 threads, generated in 2614 ms using the C++ library.
2 threads, generated in 1551 ms using the assembly library.
2 threads, generated in 2492 ms using the C++ library.
4 threads, generated in 1617 ms using the assembly library.
4 threads, generated in 2648 ms using the C++ library.
8 threads, generated in 1513 ms using the assembly library.
8 threads, generated in 2616 ms using the C++ library.
16 threads, generated in 1518 ms using the assembly library.
16 threads, generated in 2687 ms using the C++ library.
32 threads, generated in 1623 ms using the assembly library.
32 threads, generated in 2655 ms using the C++ library.
64 threads, generated in 1687 ms using the assembly library.
64 threads, generated in 2697 ms using the C++ library.

OK

5. Opis uruchamiania

Aplikacja działa poprawnie dla trybu debug i release we wszystkich konfiguracjach parametrów. Była także testowana, a jej działanie sprawdzane za pomocą debuggera i zwracanych efektów wizualnych, które zgadzały się z oczekiwaniami.

6. Podsumowanie

Projekt spełnia założenia realizując dodawanie szumów dla bibliotek c++ i asm. Atutem mojej aplikacji jest efekt wizualny powstający przy przetwarzaniu obrazów. Ponadto, udało mi się na tyle dobrze wykorzystać operacje wektorowe w asemblerze, że da się zaobserwować różnice w czasie wywołań obu bibliotek na korzyść asemblera. Im dane są większe, tym bardziej korzystne jest wykorzystanie biblioteki asm, gdyż czas jest znacząco redukowany.