ROZMYCIE GAUSSOWSKIE

Projekt z wykorzystaniem biblioteki w C++ i asemblerze

Przedstawienie i omówienie realizowanego problemu

Cyfrowa reprezentacja piksela

	ALPHA							RED				GREEN							BLUE												
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

 Kanał Alfa i składowe R,G,B przyjmują wartości od 0 do 255

- Filtrowanie obrazu polega na obliczeniu nowej wartości składowych piksela biorąc pod uwagę wartości punktów z jego otoczenia.
- Podczas obliczania wartości każdy piksel z otoczenia zostaje pomnożony przez odpowiednią wagę.
- Odpowiednie wagi zapisywane są w postaci tzw. maski.

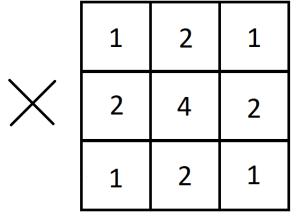
1	2	1
2	4	2
1	2	1

- W celu otrzymania nowej wartości piksela tworzymy sumę przemnożonych wartości I dzielimy ją przez normę użytego filtra
- Wartość normy flitra jest sumą wszystkich jego elementów

1	2	1
2	4	2
1	2	1

Przykład

10,10,10	15,15,15	20,20,20
25,25,25	30,30,30	35,35,35
40,40,40	45,45,45	50,50,50



10,10,10	30,30,30	20,20,20
50,50,50	120,120,120	70,70,70
40,40,40	90,90,90	50,50,50

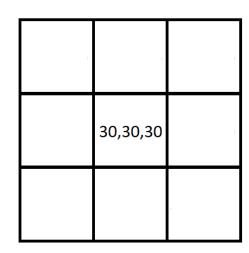
Przykład cd.

10,10,10	30,30,30	20,20,20
50,50,50	120,120,120	70,70,70
40,40,40	90,90,90	50,50,50

$$\Sigma R = 10+30+20+50+120+70+40+90+50 = 480$$

$$\Sigma G = 10+30+20+50+120+70+40+90+50 = 480$$

$$\Sigma B = 10+30+20+50+120+70+40+90+50 = 480$$



Wybrany fragment kodu

 Fragment kodu który postanowiłem zrealizować z wykorzystaniem bibliotek to obliczanie wartości piksela na podstawie jego sąsiedztwa (mnożenie przez filtr, sumowanie wartości RGB, dzielenie przez normę)

Omówienie kodu C++

```
DLL_C
                                                                                    (Global Scope)
          ⊞#include "pch.h" // use stdafx.h in Visual Studio 2017 and earlier
            #include <utility>
            #include <limits.h>
            #include "DLL C.h"
           #include <xmmintrin.h>
          □void filterImage(float* filterPointer, float* arrayPointer, int length, float filterNorm)
                __m128 XMM0; // sum of RGB |x|R|G|B|
                m128 XMM1; // currently processed pixel |x|R|G|B|
                __m128 XMM2; // current filter |x|filter|filter|
                __m128 XMM3; // norm |x|norm|norm|norm|
                XMM3 = _mm_load_ss(&filterNorm); // loading norm to the register |0|0|0|norm|
                XMM3 = _mm_unpacklo_ps(XMM3, XMM3); // unpacking norm |0|norm|0|norm|
                XMM3 = mm unpacklo ps(XMM3, XMM3); // second norm unpacking |norm|norm|norm|norm|
                XMM0 = _mm_setzero_ps(); // set all bits of XMM0 register to 0
                int offset = 0; // setting ofset
                while (length != 0)
                    XMM1 = _mm_loadu_ps(arrayPointer + (_int64)offset * 3); // moving currently processed pixel (R,G,B) to XMM1 |x|R|G|B|
                    XMM2 = _mm_loadu_ps(filterPointer + offset); // loading filter to the register
                    XMM2 = _mm_unpacklo_ps(XMM2, XMM2); // unpacking filter
                   XMM2 = _mm_unpacklo_ps(XMM2, XMM2);
                    XMM1 = _mm_mul_ps(XMM1, XMM2); // multiplying current RGB values by filter value
                    XMM0 = _mm_add_ps(XMM0, XMM1); // adding current multiplied RGB values to sum of all colors
                    offset++; // moving memory pointer
                    length -= 3; // decrementing loop counter
                XMM0 = _mm_div_ps(XMM0, XMM3); // dividing sum of colors by norm
                _mm_store_ps(arrayPointer, XMM0); // moving calculated colors back to memory
```

Omówienie kodu ASM

```
filterProc PROC
               PUSH R14
               PUSH R15
               XOR R14, R14
                                                       ; set index of color array to 0
                                                       ; set index of filter array to 0
               XOR R15, R15
                                                       ; set all bits of XMM0 register to 0
               XORPS XMM0, XMM0
               UNPCKLPS XMM3, XMM3
                                                       ; unpack norm in the register |x|norm|x|norm|
               UNPCKLPS XMM3, XMM3
                                                       ; unpack norm in the register |norm|norm|norm|
            filterLoop:
               MOVUPS XMM1, [RDX + R14]
                                                       ; move current pixel to XMM1 register |x|R|G|B|
               MOVUPS XMM2, [RCX + R15]
                                                       ; move filter to XMM1 register
               UNPCKLPS XMM2, XMM2
                                                       ; unpack current filter value in the register |x|filter|x|filter|
                                                       ; unpack current filter value in the register |filter|filter|filter|
               UNPCKLPS XMM2, XMM2
                                                       ; multiply current pixel by filter
               MULPS XMM1, XMM2
               ADDPS XMM0, XMM1
                                                       ; add multiplied values to sum of all pixels
               ADD R14, 12
                                                       ; move index of color array to the next pixel
                                                       ; move index of filter array to next filter element
               ADD R15, 4
                                                       ; decrement filter loop counter
               SUB R8, 3
                                                       ; check end of the loop
               CMP R8, 0
               JNZ filterLoop
                                                       ; if it is not the end, loop again
               DIVPS XMM0, XMM3
                                                       ; divide sum of all pixels by norm
               MOVAPS [RDX], XMM0
                                                       ; store result in memory
               POP R15
               POP R14
               RET
                                                       ; return from procedure
filterProc endp
end
```

Podsumowanie

- Zastosowanie asemblera znacznie przyśpieszyło pracę programu
- Asembler jest znacznie szybszy od c++ pomino zastosowania instrukcji wektorowych
- Asembler ma wysoki próg wejścia