

# ROZMYCIE GAUSSOWSKIE

---

Projekt z wykorzystaniem biblioteki w C++ i  
assemblerze

# 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 filtra jest sumą wszystkich jego elementów

1	2	1
2	4	2
1	2	1

$$1+2+1+2+4+2+1+2+1=16$$

# 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

 $\times$ 

1	2	1
2	4	2
1	2	1

=

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

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

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

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

$$R = 480/16 = 30$$

$$G = 480/16 = 30$$

$$B = 480/16 = 30$$

	30,30,30	

# 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)
1  #include "pch.h" // use stdafx.h in Visual Studio 2017 and earlier
2  #include <utility>
3  #include <limits.h>
4  #include "DLL_C.h"
5  #include <xmmintrin.h>
6
7  void filterImage(float* filterPointer, float* arrayPointer, int length, float filterNorm)
8  {
9
10     __m128 XMM0; // sum of RGB |x|R|G|B|
11     __m128 XMM1; // currently processed pixel |x|R|G|B|
12     __m128 XMM2; // current filter |x|filter|filter|filter|
13     __m128 XMM3; // norm |x|norm|norm|norm|
14
15     XMM3 = _mm_load_ss(&filterNorm); // loading norm to the register |0|0|0|norm|
16     XMM3 = _mm_unpacklo_ps(XMM3, XMM3); // unpacking norm |0|norm|0|norm|
17     XMM3 = _mm_unpacklo_ps(XMM3, XMM3); // second norm unpacking |norm|norm|norm|norm|
18
19     XMM0 = _mm_setzero_ps(); // set all bits of XMM0 register to 0
20     int offset = 0; // setting offset
21     while (length != 0)
22     {
23         XMM1 = _mm_loadu_ps(arrayPointer + (__int64)offset * 3); // moving currently processed pixel (R,G,B) to XMM1 |x|R|G|B|
24         XMM2 = _mm_loadu_ps(filterPointer + offset); // loading filter to the register
25         XMM2 = _mm_unpacklo_ps(XMM2, XMM2); // unpacking filter
26         XMM2 = _mm_unpacklo_ps(XMM2, XMM2);
27
28         XMM1 = _mm_mul_ps(XMM1, XMM2); // multiplying current RGB values by filter value
29         XMM0 = _mm_add_ps(XMM0, XMM1); // adding current multiplied RGB values to sum of all colors
30
31         offset++; // moving memory pointer
32         length -= 3; // decrementing loop counter
33     }
34
35     XMM0 = _mm_div_ps(XMM0, XMM3); // dividing sum of colors by norm
36     _mm_store_ps(arrayPointer, XMM0); // moving calculated colors back to memory
37 }
38
```



# Omówienie kodu ASM

```
27
28 filterProc PROC
29
30     PUSH R14
31     PUSH R15
32
33     XOR R14, R14                ; set index of color array to 0
34     XOR R15, R15                ; set index of filter array to 0
35     XORPS XMM0, XMM0            ; set all bits of XMM0 register to 0
36
37     UNPCKLPS XMM3, XMM3         ; unpack norm in the register |x|norm|x|norm|
38     UNPCKLPS XMM3, XMM3         ; unpack norm in the register |norm|norm|norm|norm|
39
40     filterLoop:
41         MOVUPS XMM1, [RDX + R14] ; move current pixel to XMM1 register |x|R|G|B|
42
43         MOVUPS XMM2, [RCX + R15] ; move filter to XMM1 register
44         UNPCKLPS XMM2, XMM2       ; unpack current filter value in the register |x|filter|x|filter|
45         UNPCKLPS XMM2, XMM2       ; unpack current filter value in the register |filter|filter|filter|filter|
46
47         MULPS XMM1, XMM2          ; multiply current pixel by filter
48         ADDPS XMM0, XMM1          ; add multiplied values to sum of all pixels
49         ADD R14, 12               ; move index of color array to the next pixel
50         ADD R15, 4               ; move index of filter array to next filter element
51         SUB R8, 3                ; decrement filter loop counter
52         CMP R8, 0               ; check end of the loop
53         JNZ filterLoop           ; if it is not the end, loop again
54
55         DIVPS XMM0, XMM3          ; divide sum of all pixels by norm
56
57         MOVAPS [RDX], XMM0        ; store result in memory
58
59         POP R15
60         POP R14
61
62         RET                      ; return from procedure
63
64 filterProc endp
65
66 end
67
```

# Podsumowanie

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