|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| logowydzialu | Instytut Informatyki Politechniki Śląskiej  Zespół Mikroinformatyki i Teorii Automatów Cyfrowych | | logoii | |
| **Rok akademicki** | **Rodzaj studiów\*: SSI/NSI/NSM** | **Przedmiot:** | **Grupa** | **Sekcja** |
| **2019/2020** | **SSI** | **Języki Asemblerowe** | **3** | **6** |
| **Data oddania:**  dd/mm/rrrr | **02.12.2019** | **Prowadzący**: | **AO** | |
| ***Raport końcowy projektu*** | | | | |
| **Temat projektu**  **Nakładanie gradientu na bitmapę.** | | | | |
| **Autor:** | Tomasz Skowron inf sem 5 | | | |

Założenia projektu

**1. Napisać główną część aplikacji konsolowej w języku C/C++.**

**2. Napisać algorytm do nakładania gradientu na bitmapę w C/C++ oraz Assemblerze.**

**3. W programie będzie możliwość wyboru funkcji (C lub Assembler).**

**4. W programie będzie możliwość wyboru ilości wykorzystanych wątków. (domyślnie maksymalna dostępna)**

**5. W programie będzie można wybrać kolor do nałożenia.**

**6. W programie dostępny będzie wybór ścieżek do plików we/wy**

Projekt zakłada stworzenie aplikacji, której główna część jest napisana w C/C++ oraz bibliotek napisanych w asm i C/C++ wykonujących takie samo przetworzenie na bloku danych wykorzystując wady i zalety odpowiednio asm i C/C++ do modyfikacji wykonania tego algorytmu. Program będzie spełniał założenia z punktów 1-6 przedstawione wyżej.

# Założenia części głównej.

Część główna programu będzie realizowała wczytywanie i zapisywanie do pliku, analizę nagłówka pliku, wczytywanie parametrów oraz zautomatyzowane testowanie czasów działania programu. Testowanie będzie polegało na wykonaniu przetwarzania bitmapy w ilości wątków od 1:64 i wygenerowanie pliku z czasami wykonania w kolejności 1 wątek, 2 wątki…,64 wątki. Program główny będzie również odpowiedzialny za dynamiczne podlinkowanie bibliotek C/C++ oraz asm.

# Założenia biblioteki w C/C++.

Biblioteka napisana w języku C/C++ będzie realizowała funkcję transformującą dane surowe w formacie RGB na takie, które będą spełniały założenia projektu. Będzie to, zatem nakładanie gradientu na bitmapę.

# Założenia biblioteki w asm.

Biblioteka napisana w języku asm będzie realizowała funkcję transformującą dane surowe w formacie RGB na takie, które będą spełniały założenia projektu. Będzie to, zatem nakładanie gradientu na bitmapę.

Biblioteki dynamiczne – opis funkcji

# Biblioteka w języku C/C++

Realizuje wyłącznie jedną funkcję, której kod przedstawiono poniżej:

DLLEXPORT void TransformBMP(unsigned char RGradient, unsigned char GGradient, unsigned char BGradient, int width, int height, void\* data)

{

int remainder = (width \* 3) % 4;

unsigned char \* temp = (unsigned char\*)data;

unsigned char CDR = RGradient;

unsigned char CDG = GGradient;

unsigned char CDB = BGradient;

int divisor = 0;

int max = ((width\*3)+remainder)\*height;

int row = 1;

int counter=0;

for (int i = 0; i < max; i += 3)

{

CDR = RGradient \* (width - divisor) / width;//factor

CDG = GGradient \* (width - divisor) / width;//factor

CDB = BGradient \* (width - divisor) / width;//factor

divisor == width ? divisor = 0 : 0;

divisor += 1;

temp[i] = (temp[i]+(CDB\*(256 - temp[i])) / 256);

temp[i + 1] = (temp[i+1]+(CDG\*(256 - temp[i + 1])) / 256);

temp[i + 2] = (temp[i + 2]+(CDR\*(256 - temp[i + 2])) / 256);

counter++;

if (counter == width)//at each row end, i+=remainder

{

counter = 0;

i += remainder;

}

}

return;

}

Funkcja ta jest idealnym przykładem na teoretyczną i praktyczną realizację gradientu, albowiem w jasny sposób przedstawia wzór określający kolejne wartości pikseli. Linie opatrzone komentarzami factor odpowiadają za obliczenie siły dokładanego koloru w zależności od jego położenia. Linie w których do zmiennej temp[…] przypisywane są wartości odpowiadają za normalizację dodawanego składnika koloru, tak aby nie doszło do przepełnienia poza zakres 0-255.

# Biblioteka napisana w asm wersja nr 1

Biblioteka w asemblerze realizuje 4 funkcje, które wspólnie odpowiadają za przetwarzanie bloku danych.

GetByte

Realizuje zadanie zgodnie z opisem w komentarzu, czyli pobiera n-ty bajt z rejestru xmm0 z zakresu 0-Fh który to zakres odczytuje ze stosu i przenosi go do rejestru al.

;Leave byte index 0h-Fh and the byte value from xmm0 will be put to al

GetByte proc byteIndex:qword

mov rbx,00FFh;make mask

mov rax, byteIndex

mov rdx,08h

sub rax,rdx

mov byteIndex,rax

jnb highbyte

add rax,rdx

mov rdx,8h

mul rdx

mov rcx,rax

mov byteIndex, rbx

shl rbx,cl

movq xmm15,xmm0

movd rax,xmm15

and rax, rbx

shr rax, cl

mov rbx,0h

movq xmm15,rbx

ret

highbyte:

movhlps xmm15,xmm0

mov rdx,8h

mul rdx

mov rcx,rax

mov byteIndex, rbx

shl rbx,cl

movd rax,xmm15

and rax, rbx

shr rax, cl

mov rbx,0h

movq xmm15,rbx

ret

GetByte endp

SetByte

Realizuje zadanie zgodnie z opisem w komentarzu, czyli pobiera n-ty bajt z zakresu 0-Fh który to zakres odczytuje ze stosu i wartość tego bajtu, następnie umieszczając go na n-tej pozycji w rejestrze xmm1

;Leave byte index 0h-Fh and the byte value, the value will replace current value in xmm1 on byteindex-th position

SetByte proc byteIndex:qword, value:qword

mov rbx,00FFh;make mask

mov rax, byteIndex

mov rdx,08h

sub rax,rdx

mov byteIndex,rax

jnb highbyte

add rax,rdx

mov rdx,8h

mul rdx

mov rcx,rax

mov byteIndex, rbx

shl rbx,cl

movq xmm15,xmm1

movd rax,xmm15

and rax, rbx

mov rbx,0h

movq xmm14,rax

xorps xmm15,xmm14

mov rax,0h

movq xmm14,rbx

movq xmm14,rax

xorps xmm15,xmm14

mov rax,value

shl rax,cl

movq xmm14,rax

orps xmm15,xmm14

movq xmm1,xmm15

mov rbx,0h

movq xmm15,rbx

ret

highbyte:

movhlps xmm15,xmm1

mov rdx,8h

mul rdx

mov rcx,rax

mov byteIndex, rbx

shl rbx,cl

movd rax,xmm15

and rax, rbx

mov rbx,0h

movq xmm14,rax

xorps xmm15,xmm14

mov rax,0h

movq xmm14,rbx

movq xmm14,rax

xorps xmm15,xmm14

mov rax,value

shl rax,cl

movq xmm14,rax

orps xmm15,xmm14

movlhps xmm1,xmm15

mov rbx,0h

movq xmm15,rbx

ret

SetByte endp

GetColorValue

Oblicza nową wartość piksela na podstawie poprzedniej. Informacje o obecnym pikselu pobiera ze stosu zgodnie z definicją funkcji. Wartość wyjściową pozostawia w rax.

;calculate new value of a pixel

;value to be returned is stored int rax

GetColorValue proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

local outValue:qword

mov rax,0h

mov outValue,rax

mov rbx,0h

mov rdx,0h

mov outValue,rax

;CDR = 200 \* (this->width - divisor) / this->width; below

mov rax,\_width

mov rbx,\_divisor

sub rax,rbx

mov rbx,0h

mov rbx,color

mul rbx

mov rbx,0h

mov rbx,\_width

mov rdx,0h

div rbx

mov outValue,rax

;temp[i]=(CDR\*(256 - temp[i])) / 256;

mov rax,100h

mov rbx,currentVal

sub rax,rbx

mov rbx,outValue

mul rbx

mov rbx,100h

div rbx

mov rbx,0h

mov rbx,currentVal

add rax,rbx

ret

;value to be returned is stored int rax

GetColorValue endp

TransformBMP

Główna funkcja biblioteki w asm. Odpowiada za kolejne wykonywanie pozostałych funkcji i przesuwanie wskaźnika na dane. Zarządza też warunkiem stopu.

TransformBMP proc RGradient:byte,GGradient:byte,BGradient:byte,\_width:dword,\_height:dword,data:qword

local returnptr: qword

local cdr:byte;current red factor

local cdg:byte;current green factor

local cdb:byte;current blue factor

local counter: qword;counter for loop

local remainder: qword;allignment remainder for read offset

local divisor: qword;algorithm divisor

local pixelsLeft: qword;pixels left in a row

local rowLoopsCount: qword;contains how many rowloops should be done, which equals to how many rows are in the data block

local registerTempPtr: qword;contains pointer to last read bytes to register

;initialize locals

mov counter, 0h

mov remainder, 0h

mov divisor,0h

mov rowLoopsCount,0h

mov registerTempPtr,0h

vzeroall

mov RGradient,cl

mov GGradient,dl

mov BGradient,r8b

mov \_width,r9d

;int remainder = (width \* 3) % 4;

mov rax,0h

mov rbx,0h

mov rcx,0h

mov rdx,0h

mov eax,\_width

mov rbx,3h

mul rbx

mov rbx,4h

div rbx

mov remainder,rdx

mov rax,0h

mov rbx,0h

mov rcx,0h

mov rdx,0h

;unsigned char CDR = RGradient;

mov al,Rgradient

mov cdr,al

;unsigned char CDG = GGradient;

mov al,GGradient

mov cdg,al

;unsigned char CDB = BGradient;

mov al,BGradient

mov cdb,al

;algorithm

;for each row

mov rax,0h

mov eax, \_height

mov rowLoopsCount, rax

rowsloop:

vzeroall

mov rax,0h

mov rbx,0h

;do stuff with those rows

mov rax, 0h

mov eax, \_width

mov pixelsLeft, rax

registerloop:

;calculate pointer offset from original data // ptr=data+(3\*divisor)+3\*((height-rowloopscount)\*width+remainder)

mov rax,divisor

mov rbx,3h

mul rbx

mov rcx,0h

mov rcx,rax;3\*divisor to rcx

mov rax,0h

mov eax,\_height

mov rbx,rowLoopsCount

sub rax,rbx

mov rbx,0h

mov ebx,\_width

mul rbx;now (height-rlc)\*width in rax

mov rbx,remainder

add rax,rbx

mov rbx,3h

mul rbx

add rax,rcx

mov rbx,rax;mov rax to rbx

mov rax,data

add rbx,rax;rbx contains pointer to current block of data to be put into xmm0

mov rax,0h ;late 3->0

add rbx,rax

movdqu xmm0, xmmword ptr [rbx]

mov registerTempPtr,rbx

;here is hardcoded algorithm for one vector analysis && edit

;------------------------0THPIXEL--------------------------------------

;--------------------------GREEN0----------------------------------------

;0th byte

mov rax,0h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,GGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,0h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------GREEN0-END-------------------------------------

;--------------------------RED1----------------------------------------

;1th byte

mov rax,1h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,RGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,1h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------RED1-END------------------------------------

;--------------------------BLUE2----------------------------------------

;2th byte

mov rax,2h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,BGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,2h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------BLUE2-END------------------------------------

;sub 1 from pixelsleft

mov rax,pixelsLeft

mov rbx,1h

sub rax,rbx

mov pixelsLeft,rax

;check if there are more pixels in this row?

cmp rax,0h

jz rowended ;if no more pixels in this row jump to next row

;divisor == width ? divisor = 0 :false;

mov rax,0h

mov rbx,0h

mov rax,divisor

mov rbx,0h

mov ebx,\_width

sub rax,rbx

cmp rax,0h

jnz noDivisorReset0;pixel n at the end of etiquete

mov rax,0h

mov divisor,rax

noDivisorReset0:;pixel n at the end of etiquete

;divisor += 1;

mov rax,divisor

mov rbx,1h

add rax,rbx

mov divisor,rax

;------------------------0THPIXEL-END----------------------------------

;------------------------1THPIXEL--------------------------------------

;--------------------------GREEN3----------------------------------------

;3th byte

mov rax,3h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,GGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,3h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------GREEN3-END-------------------------------------

;--------------------------RED4----------------------------------------

;4th byte

mov rax,4h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,RGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,4h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------RED4-END------------------------------------

;--------------------------BLUE5----------------------------------------

;5th byte

mov rax,5h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,BGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,5h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------BLUE5-END------------------------------------

;sub 1 from pixelsleft

mov rax,pixelsLeft

mov rbx,1h

sub rax,rbx

mov pixelsLeft,rax

;check if there are more pixels in this row?

cmp rax,0h

jz rowended ;if no more pixels in this row jump to next row

;divisor == width ? divisor = 0 :false;

mov rax,0h

mov rbx,0h

mov rax,divisor

mov rbx,0h

mov ebx,\_width

sub rax,rbx

cmp rax,0h

jnz noDivisorReset1;pixel n at the end of etiquete

mov rax,0h

mov divisor,rax

noDivisorReset1:;pixel n at the end of etiquete

;divisor += 1;

mov rax,divisor

mov rbx,1h

add rax,rbx

mov divisor,rax

;------------------------1THPIXEL-END----------------------------------

;------------------------2THPIXEL--------------------------------------

;--------------------------GREEN6----------------------------------------

;6th byte

mov rax,6h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,GGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,6h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------GREEN6-END-------------------------------------

;--------------------------RED7----------------------------------------

;7th byte

mov rax,7h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,RGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,7h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------RED7-END------------------------------------

;--------------------------BLUE8----------------------------------------

;8th byte

mov rax,8h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,BGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,8h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------BLUE8-END------------------------------------

;sub 1 from pixelsleft

mov rax,pixelsLeft

mov rbx,1h

sub rax,rbx

mov pixelsLeft,rax

;check if there are more pixels in this row?

cmp rax,0h

jz rowended ;if no more pixels in this row jump to next row

;divisor == width ? divisor = 0 :false;

mov rax,0h

mov rbx,0h

mov rax,divisor

mov rbx,0h

mov ebx,\_width

sub rax,rbx

cmp rax,0h

jnz noDivisorReset2;pixel n at the end of etiquete

mov rax,0h

mov divisor,rax

noDivisorReset2:;pixel n at the end of etiquete

;divisor += 1;

mov rax,divisor

mov rbx,1h

add rax,rbx

mov divisor,rax

;------------------------2THPIXEL-END----------------------------------

;------------------------3THPIXEL--------------------------------------

;--------------------------GREEN9----------------------------------------

;9th byte

mov rax,9h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,GGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,9h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------GREEN9-END-------------------------------------

;--------------------------RED10----------------------------------------

;10th byte

mov rax,0Ah;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,RGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,0Ah;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------RED10-END------------------------------------

;--------------------------BLUE11----------------------------------------

;11th byte

mov rax,0Bh;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,BGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,0Bh;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------BLUE11-END------------------------------------

;sub 1 from pixelsleft

mov rax,pixelsLeft

mov rbx,1h

sub rax,rbx

mov pixelsLeft,rax

;check if there are more pixels in this row?

cmp rax,0h

jz rowended ;if no more pixels in this row jump to next row

;divisor == width ? divisor = 0 :false;

mov rax,0h

mov rbx,0h

mov rax,divisor

mov rbx,0h

mov ebx,\_width

sub rax,rbx

cmp rax,0h

jnz noDivisorReset3;pixel n at the end of etiquete

mov rax,0h

mov divisor,rax

noDivisorReset3:;pixel n at the end of etiquete

;divisor += 1;

mov rax,divisor

mov rbx,1h

add rax,rbx

mov divisor,rax

;------------------------3THPIXEL-END----------------------------------

;------------------------4THPIXEL--------------------------------------

;--------------------------GREEN12----------------------------------------

;12th byte

mov rax,0Ch;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,GGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,0Ch;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------GREEN12-END-------------------------------------

;--------------------------RED13----------------------------------------

;13th byte

mov rax,0Dh;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,RGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,0Dh;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------RED13-END------------------------------------

;--------------------------BLUE14----------------------------------------

;14th byte

mov rax,0Eh;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,BGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,0Eh;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------BLUE14-END------------------------------------

;sub 1 from pixelsleft

mov rax,pixelsLeft

mov rbx,1h

sub rax,rbx

mov pixelsLeft,rax

;check if there are more pixels in this row?

cmp rax,0h

jz rowended ;if no more pixels in this row jump to next row

;divisor == width ? divisor = 0 :false;

mov rax,0h

mov rbx,0h

mov rax,divisor

mov rbx,0h

mov ebx,\_width

sub rax,rbx

cmp rax,0h

jnz noDivisorReset4;pixel n at the end of etiquete

mov rax,0h

mov divisor,rax

noDivisorReset4:;pixel n at the end of etiquete

;divisor += 1;

mov rax,divisor

mov rbx,1h

add rax,rbx

mov divisor,rax

;------------------------4THPIXEL-END----------------------------------

;-----------------RESTORE-15th-byte----------------------------

;15th byte

mov rax,0Fh;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

push rax;value

mov rax,0Fh;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;-----------------BYTE-15-RESTORED-NOW--------------------------

mov rbx,registerTempPtr

movdqu xmmword ptr [rbx],xmm1

jmp registerloop

rowended:

;move data a little bit so it fits remainder

mov rax,data

mov rbx,remainder

add rax,rbx

mov data,rax

;

mov rax,0h;zero the divisor

mov divisor,rax

mov rax,rowLoopsCount

mov rbx,1h

sub rax,rbx

mov rowLoopsCount,rax

cmp rax,0h

jnz rowsloop

;end of algorithm

ENDapp:

ret

TransformBMP endp

END

;-------------------------------------------------------------------------

Ogólną ideę działania TransformBMP przedstawia poniższy opis. Szczegóły są zawarte w komentarzach w kodzie.

* Pętla dla każdego rzędu
  + Pętla dla każdego segmentu (5 pikseli)
    - Obliczenie offsetu (blok danych jest jednolity; obarczony wyrównaniem do 4 bajtów, inwersją licznika {zliczamy w dół w programie}, co nie sprzyja skorzystaniu z większości typów adresowania) {ptr=data+(3\*divisor)+3\*((height-rowloopscount)\*width+remainder)+3}
    - Pobranie wektora z offsetu
    - Dla każdego z 5 pobranych do wektora wartości
      * Wyłuskanie bajtu wartości r,g,b
      * Obliczenie nowej wartości
      * Wstawienie do wektora nie zaburzając reszt
      * Zapisanie wektora do pamięci
      * Sprawdzanie warunków końca pętli rejestowej
      * Sprawdzenie warunku końca danych.

TransformBMP opiera swoje działanie w większości na rejestrach rax, rbx, rcx, rdx, r8, r9, xmm0,xmm1, pozostałe rejestry są potrzebne do zachowania wartości tymczasowych. Ponadto program używa kilku zmiennych przechowujących wartości tymczasowe liczników i zapamiętujące wartości kolorów, wysokości, szerokości.

Wykorzystane instrukcje wektorowe:

* movq
* movhlps
* movlhps
* xorps
* orps
* vzeroall
* movdqu

# Biblioteka napisana w asm wersja nr 2

Funkcje SetByte, GetByte oraz GetColovalue pozostają niezmienione.

Zmiany zostały dokonane w funkcji TansformBMP.

Polegały na zamianie idei wyciągania kolejnych pikseli z wektorów i ich analizy, po której następowało przygotowanie nowego wektora i jego zapis.

W wersji 2-giej zastosowano operacje równoległe, które w założeniu powinny przyśpieszyć działanie programu. Algorytm pozostaje niezmieniony, a jedynie zmodyfikowany by w konkretnych momentach programu wykonywał równolegle od 1 do 16 operacji. Niektóre operacje muszą być przeprowadzane osobno, jak przygotowanie wektora xmm11, który zawiera kolejne wartości zależne od położenia piksela na obrazie.

Szczegółowe omówienie realizacji algorytmu zajęłoby zbyt wiele czasu, dlatego należy się skupić na idei jaka przyświecała zrównolegleniu tego procesu. Na przykładzie algorytmu napisanego w C/C++ wyznaczamy, że

*temp[i]=temp[i]+(Gi\*((width-divisori)/width)\*(256-temp[i]))/256*

gdzie i oznacza i-ty bajt. Gi jest to odpowiedni gradient dla danego bajtu. divisori jest to odpowiedni dzielnik dla danych bajtów z danego piksela. Algorytm umieszcza w sposób szeregowy część *(Gi\*((width-divisori)/width)* w rejestrze xmm12, część *(256-temp[i])* w rejestrze xmm11, a następnie dokonuje wymnożenia (po zastosowaniu wcześniej konwersji byte to dword). Po wymnożeniu wykonywana jest konwersja z dword na single-precision floating point, po której następuje dzielenie przez 256. Wynik tego działania, po konwersji w dół (manualnej ze względu na brak rozszerzenia AVX512) jest dodawany do poprzedniej wartości analizowanego obszaru pamięci i zapisywany. W trakcie mnożenia przetwarzane są 4 bajty na raz, zaś przy dodawaniu 16.

Kod programu został umieszczony poniżej (bez funkcji niezmienionych):

TransformBMP proc RGradient:byte,GGradient:byte,BGradient:byte,\_width:dword,\_height:dword,data:qword

local returnptr: qword

local cdr:byte;current red factor

local cdg:byte;current green factor

local cdb:byte;current blue factor

local counter: qword;counter for loop

local remainder: qword;allignment remainder for read offset

local divisor: qword;algorithm divisor

local pixelsLeft: qword;pixels left in a row

local rowLoopsCount: qword;contains how many rowloops should be done, which equals to how many rows are in the data block

local registerTempPtr: qword;contains pointer to last read bytes to register

local maxChar: byte

;initialize locals

mov counter, 0h

mov remainder, 0h

mov divisor,0h

mov rowLoopsCount,0h

mov registerTempPtr,0h

mov maxChar,00FFh

vzeroall

mov RGradient,cl

mov GGradient,dl

mov BGradient,r8b

mov \_width,r9d

;int remainder = (width \* 3) % 4;

mov rax,0h

mov rbx,0h

mov rcx,0h

mov rdx,0h

mov eax,\_width

mov rbx,3h

mul rbx

mov rbx,4h

div rbx

mov remainder,rdx

mov rax,0h

mov rbx,0h

mov rcx,0h

mov rdx,0h

;unsigned char CDR = RGradient;

mov al,Rgradient

mov cdr,al

;unsigned char CDG = GGradient;

mov al,GGradient

mov cdg,al

;unsigned char CDB = BGradient;

mov al,BGradient

mov cdb,al

;algorithm

;for each row

mov rax,0h

mov eax, \_height

mov rowLoopsCount, rax

rowsloop:

vzeroall

mov rax,0h

mov rbx,0h

;do stuff with those rows

mov rax, 0h

mov eax, \_width

mov pixelsLeft, rax

registerloop:

;calculate pointer offset from original data // ptr=data+(3\*divisor)+3\*((height-rowloopscount)\*width+remainder)

mov rax,divisor

mov rbx,3h

mul rbx

mov rcx,0h

mov rcx,rax;3\*divisor to rcx

mov rax,0h

mov eax,\_height

mov rbx,rowLoopsCount

sub rax,rbx

mov rbx,0h

mov ebx,\_width

mul rbx;now (height-rlc)\*width in rax

mov rbx,remainder

add rax,rbx

mov rbx,3h

mul rbx

add rax,rcx

mov rbx,rax;mov rax to rbx

mov rax,data

add rbx,rax;rbx contains pointer to current block of data to be put into xmm0

mov rax,0h ;late 3->0

add rbx,rax

movdqu xmm0, xmmword ptr [rbx]

mov registerTempPtr,rbx

;here is hardcoded algorithm for one vector analysis && edit

mov rax,pixelsLeft

sub rax,5h

jb special

;save last pixel

mov rax,0FH

push rax

call GetByte

pop rbx

mov r8b,al;LAST BYTE IN r8b

movdqu xmm13,xmm1;Gradients to xmm13

xorps xmm1,xmm1

;prepare xmm register with: RG\*(width-divisor)\*/width

;px1

mov rax,0h

mov eax,\_width

mov rbx,divisor

sub rax,rbx

mov r9,rax;save (width-divisor)

mov rbx,0h

mov bl,RGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,0h

push rbx

call SetByte

pop rbx

pop rbx

mov rax,r9;restore (width-divisor)

mov rbx,0h

mov bl,GGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,1h

push rbx

call SetByte

pop rbx

pop rbx

mov rax,r9;restore (width-divisor)

mov rbx,0h

mov bl,BGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,2h

push rbx

call SetByte

pop rbx

pop rbx

;take care of divisor

mov rax,divisor;increment divisor

inc rax;increment divisor

mov divisor,rax;increment divisor

;px2

mov rax,0h

mov eax,\_width

mov rbx,divisor

sub rax,rbx

mov r9,rax;save (width-divisor)

mov rbx,0h

mov bl,RGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,3h

push rbx

call SetByte

pop rbx

pop rbx

mov rax,r9;restore (width-divisor)

mov rbx,0h

mov bl,GGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,4h

push rbx

call SetByte

pop rbx

pop rbx

mov rax,r9;restore (width-divisor)

mov rbx,0h

mov bl,BGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,5h

push rbx

call SetByte

pop rbx

pop rbx

;take care of divisor

mov rax,divisor;increment divisor

inc rax;increment divisor

mov divisor,rax;increment divisor

;px3

mov rax,0h

mov eax,\_width

mov rbx,divisor

sub rax,rbx

mov r9,rax;save (width-divisor)

mov rbx,0h

mov bl,RGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,6h

push rbx

call SetByte

pop rbx

pop rbx

mov rax,r9;restore (width-divisor)

mov rbx,0h

mov bl,GGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,7h

push rbx

call SetByte

pop rbx

pop rbx

mov rax,r9;restore (width-divisor)

mov rbx,0h

mov bl,BGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,8h

push rbx

call SetByte

pop rbx

pop rbx

;take care of divisor

mov rax,divisor;increment divisor

inc rax;increment divisor

mov divisor,rax;increment divisor

;px4

mov rax,0h

mov eax,\_width

mov rbx,divisor

sub rax,rbx

mov r9,rax;save (width-divisor)

mov rbx,0h

mov bl,RGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,9h

push rbx

call SetByte

pop rbx

pop rbx

mov rax,r9;restore (width-divisor)

mov rbx,0h

mov bl,GGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,0Ah

push rbx

call SetByte

pop rbx

pop rbx

mov rax,r9;restore (width-divisor)

mov rbx,0h

mov bl,BGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,0Bh

push rbx

call SetByte

pop rbx

pop rbx

;take care of divisor

mov rax,divisor;increment divisor

inc rax;increment divisor

mov divisor,rax;increment divisor

;px5

mov rax,0h

mov eax,\_width

mov rbx,divisor

sub rax,rbx

mov r9,rax;save (width-divisor)

mov rbx,0h

mov bl,RGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,0Ch

push rbx

call SetByte

pop rbx

pop rbx

mov rax,r9;restore (width-divisor)

mov rbx,0h

mov bl,GGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,0Dh

push rbx

call SetByte

pop rbx

pop rbx

mov rax,r9;restore (width-divisor)

mov rbx,0h

mov bl,BGradient

mul rbx

mov rbx,0h

mov ebx,\_width

div rbx

push rax ;RG\*(push width-divisor)/width

;setbyte push value, then push index

mov rbx,0Eh

push rbx

call SetByte

pop rbx

pop rbx

;take care of divisor

mov rax,divisor;increment divisor

inc rax;increment divisor

mov divisor,rax;increment divisor

movdqu xmm12,xmm1 ;(width-height)/width to xmm12

xorps xmm1,xmm1

;prepare xmm register with (256-temp[i])

VPBROADCASTB xmm1,maxChar

vpsubb xmm2,xmm1,xmm0

movdqu xmm11,xmm2

xorps xmm2,xmm2

xorps xmm1,xmm1

;256-temp[i] in xmm11

;1st 4 bytes

vpmovzxbd xmm1,xmm11

vpmovzxbd xmm2,xmm12

vpmulld xmm2,xmm1,xmm2;in xmm2 there is (RG\*(width-divisor)/width)\*(256-temp[i]) for 1st 4 bytes

VPBROADCASTB xmm9,maxChar

vpmovzxbd xmm1,xmm9

VCVTDQ2PS xmm3,xmm1;to single precision

VCVTDQ2PS xmm4,xmm2

vdivps xmm4,xmm4,xmm3;divide all stuff by 256

VCVTPS2DQ xmm5,xmm4;back to integer

;convert dwords to bytes

movq xmm6,xmm5

psrldq xmm6,1h

pslldq xmm6,1h

pxor xmm5,xmm6

psrldq xmm6,3h

por xmm5,xmm6

movhlps xmm6,xmm5

movq xmm7,xmm6

pslldq xmm6,0Ch

psrldq xmm6,0Ch

pxor xmm7,xmm6

pslldq xmm6,2h

por xmm5,xmm6

psrldq xmm7,1h

por xmm5,xmm7

movq xmm10,xmm5;move 1st 4 bytes to xmm10

pxor xmm1,xmm1

pxor xmm2,xmm2

pxor xmm3,xmm3

pxor xmm4,xmm4

pxor xmm5,xmm5

pxor xmm6,xmm6

pxor xmm7,xmm7

pxor xmm9,xmm9

;2nd 4 bytes

psrldq xmm11,4h

psrldq xmm12,4h

psrldq xmm13,4h

vpmovzxbd xmm1,xmm11

vpmovzxbd xmm2,xmm12

vpmulld xmm2,xmm1,xmm2;in xmm2 there is (RG\*(width-divisor)/width)\*(256-temp[i]) for 1st 4 bytes

VPBROADCASTB xmm9,maxChar

vpmovzxbd xmm1,xmm9

VCVTDQ2PS xmm3,xmm1;to single precision

VCVTDQ2PS xmm4,xmm2

vdivps xmm4,xmm4,xmm3;divide all stuff by 256

VCVTPS2DQ xmm5,xmm4;back to integer

;convert dwords to bytes

movq xmm6,xmm5

psrldq xmm6,1h

pslldq xmm6,1h

pxor xmm5,xmm6

psrldq xmm6,3h

por xmm5,xmm6

movhlps xmm6,xmm5

movq xmm7,xmm6

pslldq xmm6,0Ch

psrldq xmm6,0Ch

pxor xmm7,xmm6

pslldq xmm6,2h

por xmm5,xmm6

psrldq xmm7,1h

por xmm5,xmm7

pslldq xmm5,4h

pslldq xmm5,8h

psrldq xmm5,8h

por xmm10,xmm5;move 1st 4 bytes to xmm10

pxor xmm1,xmm1

pxor xmm2,xmm2

pxor xmm3,xmm3

pxor xmm4,xmm4

pxor xmm5,xmm5

pxor xmm6,xmm6

pxor xmm7,xmm7

pxor xmm9,xmm9

;3rd 4 bytes

psrldq xmm11,4h

psrldq xmm12,4h

psrldq xmm13,4h

vpmovzxbd xmm1,xmm11

vpmovzxbd xmm2,xmm12

vpmulld xmm2,xmm1,xmm2;in xmm2 there is (RG\*(width-divisor)/width)\*(256-temp[i]) for 1st 4 bytes

VPBROADCASTB xmm9,maxChar

vpmovzxbd xmm1,xmm9

VCVTDQ2PS xmm3,xmm1;to single precision

VCVTDQ2PS xmm4,xmm2

vdivps xmm4,xmm4,xmm3;divide all stuff by 256

VCVTPS2DQ xmm5,xmm4;back to integer

;convert dwords to bytes

movq xmm6,xmm5

psrldq xmm6,1h

pslldq xmm6,1h

pxor xmm5,xmm6

psrldq xmm6,3h

por xmm5,xmm6

movhlps xmm6,xmm5

movq xmm7,xmm6

pslldq xmm6,0Ch

psrldq xmm6,0Ch

pxor xmm7,xmm6

pslldq xmm6,2h

por xmm5,xmm6

psrldq xmm7,1h

por xmm5,xmm7

pslldq xmm5,4h

pslldq xmm5,8h

psrldq xmm5,4h

por xmm10,xmm5;move 1st 4 bytes to xmm10

pxor xmm1,xmm1

pxor xmm2,xmm2

pxor xmm3,xmm3

pxor xmm4,xmm4

pxor xmm5,xmm5

pxor xmm6,xmm6

pxor xmm7,xmm7

pxor xmm9,xmm9

;4th 4 bytes

psrldq xmm11,4h

psrldq xmm12,4h

psrldq xmm13,4h

vpmovzxbd xmm1,xmm11

vpmovzxbd xmm2,xmm12

vpmulld xmm2,xmm1,xmm2;in xmm2 there is (RG\*(width-divisor)/width)\*(256-temp[i]) for 1st 4 bytes

VPBROADCASTB xmm9,maxChar

vpmovzxbd xmm1,xmm9

VCVTDQ2PS xmm3,xmm1;to single precision

VCVTDQ2PS xmm4,xmm2

vdivps xmm4,xmm4,xmm3;divide all stuff by 256

VCVTPS2DQ xmm5,xmm4;back to integer

;convert dwords to bytes

movq xmm6,xmm5

psrldq xmm6,1h

pslldq xmm6,1h

pxor xmm5,xmm6

psrldq xmm6,3h

por xmm5,xmm6

movhlps xmm6,xmm5

movq xmm7,xmm6

pslldq xmm6,0Ch

psrldq xmm6,0Ch

pxor xmm7,xmm6

pslldq xmm6,2h

por xmm5,xmm6

psrldq xmm7,1h

por xmm5,xmm7

pslldq xmm5,4h

pslldq xmm5,8h

por xmm10,xmm5;move 1st 4 bytes to xmm10

pxor xmm1,xmm1

pxor xmm2,xmm2

pxor xmm3,xmm3

pxor xmm4,xmm4

pxor xmm5,xmm5

pxor xmm6,xmm6

pxor xmm7,xmm7

pxor xmm9,xmm9

vpaddb xmm1,xmm10,xmm0

mov rax,0h

mov al,r8b

push rax

mov rax,0Fh

push rax

call SetByte

pop rbx

pop rbx

pxor xmm10,xmm10

pxor xmm11,xmm11

pxor xmm12,xmm12

pxor xmm13,xmm13

;on register loop end

;sub 5 pixels from pixels left

mov rax,0h

mov rax, pixelsLeft

mov rbx,5h

sub rax,rbx

mov pixelsLeft,rax

;check if there are more pixels in this row?

cmp rax,0h

jna rowended ;if no more pixels in this row jump to next row

;divisor == width ? divisor = 0 :false;

mov rax,0h

mov rbx,0h

mov rax,divisor

mov rbx,0h

mov ebx,\_width

sub rax,rbx

cmp rax,0h

jnz noDivisorReset;pixel n at the end of etiquete

mov rax,0h

mov divisor,rax

noDivisorReset:;pixel n at the end of etiquete

mov rbx,registerTempPtr

movdqu xmmword ptr [rbx],xmm1

jmp registerloop

special:

;------------------------0THPIXEL--------------------------------------

;--------------------------GREEN0----------------------------------------

;0th byte

mov rax,0h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,GGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,0h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------GREEN0-END-------------------------------------

;--------------------------RED1----------------------------------------

;1th byte

mov rax,1h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,RGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,1h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------RED1-END------------------------------------

;--------------------------BLUE2----------------------------------------

;2th byte

mov rax,2h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,BGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,2h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------BLUE2-END------------------------------------

;sub 1 from pixelsleft

mov rax,pixelsLeft

mov rbx,1h

sub rax,rbx

mov pixelsLeft,rax

;check if there are more pixels in this row?

cmp rax,0h

jz rowended ;if no more pixels in this row jump to next row

;divisor == width ? divisor = 0 :false;

mov rax,0h

mov rbx,0h

mov rax,divisor

mov rbx,0h

mov ebx,\_width

sub rax,rbx

cmp rax,0h

jnz noDivisorReset0;pixel n at the end of etiquete

mov rax,0h

mov divisor,rax

noDivisorReset0:;pixel n at the end of etiquete

;divisor += 1;

mov rax,divisor

mov rbx,1h

add rax,rbx

mov divisor,rax

;------------------------0THPIXEL-END----------------------------------

;------------------------1THPIXEL--------------------------------------

;--------------------------GREEN3----------------------------------------

;3th byte

mov rax,3h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,GGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,3h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------GREEN3-END-------------------------------------

;--------------------------RED4----------------------------------------

;4th byte

mov rax,4h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,RGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,4h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------RED4-END------------------------------------

;--------------------------BLUE5----------------------------------------

;5th byte

mov rax,5h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,BGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,5h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------BLUE5-END------------------------------------

;sub 1 from pixelsleft

mov rax,pixelsLeft

mov rbx,1h

sub rax,rbx

mov pixelsLeft,rax

;check if there are more pixels in this row?

cmp rax,0h

jz rowended ;if no more pixels in this row jump to next row

;divisor == width ? divisor = 0 :false;

mov rax,0h

mov rbx,0h

mov rax,divisor

mov rbx,0h

mov ebx,\_width

sub rax,rbx

cmp rax,0h

jnz noDivisorReset1;pixel n at the end of etiquete

mov rax,0h

mov divisor,rax

noDivisorReset1:;pixel n at the end of etiquete

;divisor += 1;

mov rax,divisor

mov rbx,1h

add rax,rbx

mov divisor,rax

;------------------------1THPIXEL-END----------------------------------

;------------------------2THPIXEL--------------------------------------

;--------------------------GREEN6----------------------------------------

;6th byte

mov rax,6h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,GGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,6h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------GREEN6-END-------------------------------------

;--------------------------RED7----------------------------------------

;7th byte

mov rax,7h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,RGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,7h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------RED7-END------------------------------------

;--------------------------BLUE8----------------------------------------

;8th byte

mov rax,8h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,BGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,8h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------BLUE8-END------------------------------------

;sub 1 from pixelsleft

mov rax,pixelsLeft

mov rbx,1h

sub rax,rbx

mov pixelsLeft,rax

;check if there are more pixels in this row?

cmp rax,0h

jz rowended ;if no more pixels in this row jump to next row

;divisor == width ? divisor = 0 :false;

mov rax,0h

mov rbx,0h

mov rax,divisor

mov rbx,0h

mov ebx,\_width

sub rax,rbx

cmp rax,0h

jnz noDivisorReset2;pixel n at the end of etiquete

mov rax,0h

mov divisor,rax

noDivisorReset2:;pixel n at the end of etiquete

;divisor += 1;

mov rax,divisor

mov rbx,1h

add rax,rbx

mov divisor,rax

;------------------------2THPIXEL-END----------------------------------

;------------------------3THPIXEL--------------------------------------

;--------------------------GREEN9----------------------------------------

;9th byte

mov rax,9h;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,GGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,9h;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------GREEN9-END-------------------------------------

;--------------------------RED10----------------------------------------

;10th byte

mov rax,0Ah;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,RGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,0Ah;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------RED10-END------------------------------------

;--------------------------BLUE11----------------------------------------

;11th byte

mov rax,0Bh;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,BGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,0Bh;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------BLUE11-END------------------------------------

;sub 1 from pixelsleft

mov rax,pixelsLeft

mov rbx,1h

sub rax,rbx

mov pixelsLeft,rax

;check if there are more pixels in this row?

cmp rax,0h

jz rowended ;if no more pixels in this row jump to next row

;divisor == width ? divisor = 0 :false;

mov rax,0h

mov rbx,0h

mov rax,divisor

mov rbx,0h

mov ebx,\_width

sub rax,rbx

cmp rax,0h

jnz noDivisorReset3;pixel n at the end of etiquete

mov rax,0h

mov divisor,rax

noDivisorReset3:;pixel n at the end of etiquete

;divisor += 1;

mov rax,divisor

mov rbx,1h

add rax,rbx

mov divisor,rax

;------------------------3THPIXEL-END----------------------------------

;------------------------4THPIXEL--------------------------------------

;--------------------------GREEN12----------------------------------------

;12th byte

mov rax,0Ch;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,GGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,0Ch;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------GREEN12-END-------------------------------------

;--------------------------RED13----------------------------------------

;13th byte

mov rax,0Dh;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,RGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,0Dh;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------RED13-END------------------------------------

;--------------------------BLUE14----------------------------------------

;14th byte

mov rax,0Eh;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

mov rbx,0h

;get pixel.color value

mov rbx,rax;hide currentValue in rbx

;push needed stuff onto stack

push rax;currentVal got from getByte

mov rax,0h

mov al,BGradient;color factor now

push rax;color

mov rax,divisor

push rax;divisor

mov rax,0h

mov eax,\_width

push rax;\_width

;get new value

call GetColorValue;proc \_width:qword,\_divisor:qword,color:qword,currentVal:qword

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

push rax;value

mov rax,0Eh;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;--------------------------BLUE14-END------------------------------------

;sub 1 from pixelsleft

mov rax,pixelsLeft

mov rbx,1h

sub rax,rbx

mov pixelsLeft,rax

;check if there are more pixels in this row?

cmp rax,0h

jz rowended ;if no more pixels in this row jump to next row

;divisor == width ? divisor = 0 :false;

mov rax,0h

mov rbx,0h

mov rax,divisor

mov rbx,0h

mov ebx,\_width

sub rax,rbx

cmp rax,0h

jnz noDivisorReset4;pixel n at the end of etiquete

mov rax,0h

mov divisor,rax

noDivisorReset4:;pixel n at the end of etiquete

;divisor += 1;

mov rax,divisor

mov rbx,1h

add rax,rbx

mov divisor,rax

;------------------------4THPIXEL-END----------------------------------

;-----------------RESTORE-15th-byte----------------------------

;15th byte

mov rax,0Fh;nth byte

push rax

call GetByte;proc byteIndex:qword

pop rbx;cleanup

push rax;value

mov rax,0Fh;nth byte

push rax

call SetByte;byteIndex:qword, value:qword

pop rbx;cleanup

pop rbx;cleanup

mov rbx,0h

;-----------------BYTE-15-RESTORED-NOW--------------------------

mov rbx,registerTempPtr

movdqu xmmword ptr [rbx],xmm1

rowended:

;move data a little bit so it fits remainder

mov rax,data

mov rbx,remainder

add rax,rbx

mov data,rax

;

mov rax,0h;zero the divisor

mov divisor,rax

mov rax,rowLoopsCount

mov rbx,1h

sub rax,rbx

mov rowLoopsCount,rax

cmp rax,0h

jnz rowsloop

;end of algorithm

ENDapp:

ret

TransformBMP endp

END

;-------------------------------------------------------------------------

W tej wersji funkcji transformBMP (poza tymi z wersji nr 1) użyto następujących instrukcji wektorowych:

* vpbroadcastb
* vpsubb
* movdqu
* xorps
* vpmovzxbd
* vpmulld
* vcvtdq2ps
* vdivps
* pslrdq
* pslldq

# Opis struktury danych wejściowych/testowych

Program działa na danych zapisanych w formacie .bmp i zmienia wyłącznie surowe dane, nagłówek pozostaje nienaruszony.

Poniżej został zaprezentowany kod odpowiadający za odczyt nagłówka ze struktury BMP:

////https://solarianprogrammer.com/2018/11/19/cpp-reading-writing-bmp-images/

#pragma pack(push,1)

struct BMPFileHeader {

uint16\_t file\_type{ 0 };

uint32\_t file\_size{ 0 };

uint16\_t reserved1{ 0 };

uint16\_t reserved2{ 0 };

uint32\_t offset\_data{ 0 };

};

#pragma pack(pop)

#pragma pack(push,1)

struct BMPInfoHeader {

uint32\_t size{ 0 };

int32\_t width{ 0 };

int32\_t height{ 0 };

uint16\_t planes{ 1 };

uint16\_t bit\_count{ 0 };

uint32\_t compression{ 0 };

uint32\_t size\_image{ 0 };

int32\_t x\_pixels\_per\_meter{ 0 };

int32\_t y\_pixels\_per\_meter{ 0 };

uint32\_t colors\_used{ 0 };

uint32\_t colors\_important{ 0 };

};

#pragma pack(pop)

#pragma pack(push,1)

struct BMPColorHeader {

uint32\_t red\_mask{ 0x00ff0000 }; // Bit mask for the red channel

uint32\_t green\_mask{ 0x0000ff00 }; // Bit mask for the green channel

uint32\_t blue\_mask{ 0x000000ff }; // Bit mask for the blue channel

uint32\_t alpha\_mask{ 0xff000000 }; // Bit mask for the alpha channel

uint32\_t color\_space\_type{ 0x73524742 }; // Default "sRGB" (0x73524742)

uint32\_t unused[16]{ 0 }; // Unused data for sRGB color space

};

#pragma pack(pop)

Ten fragment kodu został stworzony na podstawie:

////https://solarianprogrammer.com/2018/11/19/cpp-reading-writing-bmp-images/

# Opis parametrów programu.

Program przyjmuje pliki wejściowe i wyjściowe za pomocą parametrów z linii poleceń. Nie zostały zaimplementowane pełne zabezpieczenia przed niepoprawnymi parametrami. Program ogranicza jedynie zakres ilości wątków. Program nie odpowiada za niepoprawnie podane ścieżki do plików, ani nieprawidłowe wartości kolorów RGB do nałożenia.

Program przyjmuje następujące parametry, po których musi nastąpić zawsze wartość w postaci liczby naturalnej lub ciągu znaków:

-act -> parametr po którym zaleca się podać wartość 0. Jeśli został zdefiniowany jako pierwszy to program nie wykona testów szybkości działania opartych o kolejne zwiększanie liczby wątków dla przetwarzania bitmapy. Wykona za równo testy dla asm, jak i C/C++ i zapisze do plików kolejne wyniki przy rosnącej liczbie wątków.

-t -> parametr po którym należy podać wartość z zakresu 1-64, który określi ile wątków chcemy użyć. Przy komputerze na którym sprzętowo udostępniono 8 wątków każda wartość powyżej ograniczenia sprzętowego była bardziej kosztowna niż wywołanie 8 wątków. Wykres zostanie zaprezentowany w dalszej części sprawozdania.

-r -> parametr oczekuje wartość R gradientu, zakres od 0-255

-g -> parametr oczekuje wartość G gradientu, zakres od 0-255

-b -> parametr oczekuje wartość B gradientu, zakres od 0-255

-i -> parametr oczekuje podania ścieżki do pliku wejściowego

-o -> parametr oczekuje podania ścieżki do pliku wyjściowego

-A1C0 -> parametr oczekuje podania wartości 1 lub 0 w zależności od tego czy chcemy wykonać dllkę w asm, czy C/C++. Dla asm należy podać 1, dla C należy podać 0.

# Wyniki pomiarów czasowych wykonania programu.

# Wersja 1

Dla języka C/C++ otrzymano. Wyniki są kolejno dla 1, 2…. 64 wątków.

0.107422

0.054526

0.051868

0.035843

0.031509

0.032510

0.041651

0.043182

0.035951

0.041382

0.036535

0.032926

0.038313

0.033744

0.036129

0.041908

0.036636

0.036393

0.041000

0.036125

0.038876

0.040416

0.040328

0.035047

0.039577

0.039221

0.039622

0.058646

0.040966

0.038382

0.040709

0.047581

0.040607

0.042031

0.047019

0.044185

0.041097

0.048217

0.048136

0.046688

0.045976

0.053524

0.047913

0.047774

0.045885

0.050474

0.056143

0.051749

0.060375

0.053184

0.056333

0.061288

0.057547

0.058326

0.072884

0.060980

0.059547

0.068253

0.066411

0.056842

0.056205

0.059920

0.057739

0.062121

Dla języka asm otrzymano. Wyniki są kolejno dla 1, 2…. 64 wątków.

0.481928

0.251234

0.167308

0.158923

0.139788

0.122606

0.117521

0.112771

0.132386

0.137262

0.132558

0.130683

0.152552

0.126329

0.121700

0.123514

0.129431

0.135764

0.124095

0.166609

0.129063

0.121898

0.126601

0.120049

0.149467

0.211417

0.156639

0.147438

0.135582

0.125199

0.147939

0.159808

0.132574

0.151895

0.156100

0.135021

0.134181

0.130754

0.131938

0.139337

0.147971

0.145254

0.135937

0.129829

0.131912

0.145359

0.153391

0.143006

0.143694

0.144520

0.148220

0.190979

0.171175

0.151425

0.171177

0.216911

0.209241

0.159551

0.181883

0.163580

0.169020

0.144439

0.169060

0.160245

Wykresy przedstawione poniżej obrazują prędkość działania obu algorytmów.

Przy różnych wielkościach bitmap trend zmian był zachowany i analogiczny do zaprezentowanych na wykresach.

# Wersja 2

Po wykonaniu programu w wersji nr 2 otrzymano następującą tabelę dla wesji debug:

|  |  |  |
| --- | --- | --- |
| Liczba wątków | Assembler | C/C++ |
| 1 | 0,254251 | 0,162081 |
| 2 | 0,132222 | 0,075027 |
| 3 | 0,093862 | 0,04673 |
| 4 | 0,070984 | 0,042482 |
| 5 | 0,076463 | 0,045185 |
| 6 | 0,068954 | 0,034969 |
| 7 | 0,07543 | 0,033113 |
| 8 | 0,057641 | 0,056621 |
| 9 | 0,073287 | 0,050463 |
| 10 | 0,075002 | 0,046579 |
| 11 | 0,069574 | 0,043552 |
| 12 | 0,070161 | 0,043194 |
| 13 | 0,071799 | 0,053424 |
| 14 | 0,073257 | 0,040763 |
| 15 | 0,077107 | 0,040831 |
| 16 | 0,069083 | 0,044694 |
| 17 | 0,083856 | 0,045841 |
| 18 | 0,074343 | 0,054343 |
| 19 | 0,103412 | 0,041049 |
| 20 | 0,073852 | 0,046142 |
| 21 | 0,081746 | 0,053745 |
| 22 | 0,073636 | 0,053682 |
| 23 | 0,091851 | 0,047255 |
| 24 | 0,085403 | 0,048329 |
| 25 | 0,074806 | 0,046584 |
| 26 | 0,07485 | 0,055905 |
| 27 | 0,084234 | 0,055193 |
| 28 | 0,079688 | 0,064092 |
| 29 | 0,079344 | 0,064009 |
| 30 | 0,081877 | 0,073208 |
| 31 | 0,072682 | 0,044171 |
| 32 | 0,084682 | 0,051879 |
| 33 | 0,078007 | 0,056495 |
| 34 | 0,086721 | 0,053352 |
| 35 | 0,080308 | 0,049731 |
| 36 | 0,085613 | 0,058809 |
| 37 | 0,08574 | 0,058371 |
| 38 | 0,09168 | 0,065328 |
| 39 | 0,085616 | 0,05435 |
| 40 | 0,09553 | 0,061093 |
| 41 | 0,086262 | 0,056183 |
| 42 | 0,093595 | 0,063903 |
| 43 | 0,0927 | 0,054446 |
| 44 | 0,08219 | 0,061928 |
| 45 | 0,092299 | 0,060411 |
| 46 | 0,086358 | 0,066648 |
| 47 | 0,091842 | 0,066513 |
| 48 | 0,090798 | 0,063343 |
| 49 | 0,097143 | 0,068039 |
| 50 | 0,090592 | 0,06404 |
| 51 | 0,104196 | 0,06167 |
| 52 | 0,097358 | 0,063444 |
| 53 | 0,087069 | 0,060124 |
| 54 | 0,107313 | 0,071159 |
| 55 | 0,098778 | 0,067478 |
| 56 | 0,10013 | 0,069394 |
| 57 | 0,097121 | 0,06466 |
| 58 | 0,102342 | 0,070988 |
| 59 | 0,095649 | 0,068506 |
| 60 | 0,090776 | 0,073874 |
| 61 | 0,115218 | 0,076481 |
| 62 | 0,099636 | 0,076259 |
| 63 | 0,106331 | 0,064535 |
| 64 | 0,10604 | 0,073579 |

Dla wersji release otrzymano:

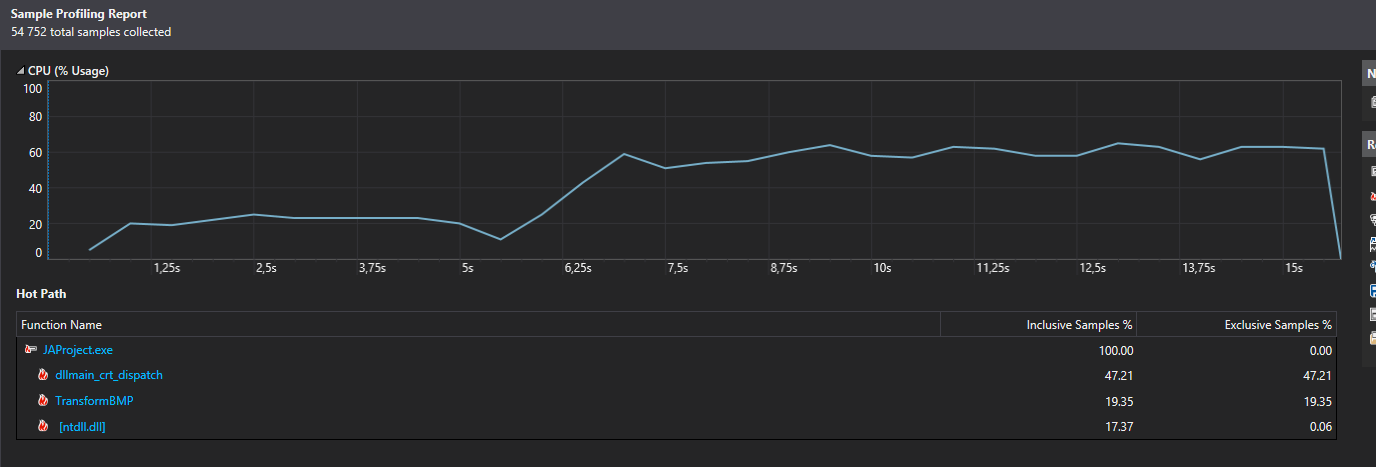
|  |  |  |
| --- | --- | --- |
| Liczba wątków | Assembler | C/C++ |
| 1 | 0,258955 | 0,084063 |
| 2 | 0,144105 | 0,05151 |
| 3 | 0,109777 | 0,046365 |
| 4 | 0,089808 | 0,03256 |
| 5 | 0,074528 | 0,031682 |
| 6 | 0,067422 | 0,027553 |
| 7 | 0,061757 | 0,027108 |
| 8 | 0,057326 | 0,042997 |
| 9 | 0,073341 | 0,034636 |
| 10 | 0,07223 | 0,032209 |
| 11 | 0,077984 | 0,03362 |
| 12 | 0,074744 | 0,035102 |
| 13 | 0,071162 | 0,036148 |
| 14 | 0,065792 | 0,033818 |
| 15 | 0,06358 | 0,037697 |
| 16 | 0,061185 | 0,034009 |
| 17 | 0,072489 | 0,036324 |
| 18 | 0,07363 | 0,036712 |
| 19 | 0,073571 | 0,037527 |
| 20 | 0,076348 | 0,044293 |
| 21 | 0,073524 | 0,035588 |
| 22 | 0,075282 | 0,04093 |
| 23 | 0,070535 | 0,041093 |
| 24 | 0,077959 | 0,035801 |
| 25 | 0,072629 | 0,039133 |
| 26 | 0,078225 | 0,03904 |
| 27 | 0,075127 | 0,036301 |
| 28 | 0,075108 | 0,043833 |
| 29 | 0,079806 | 0,042142 |
| 30 | 0,079229 | 0,041353 |
| 31 | 0,081729 | 0,041614 |
| 32 | 0,079907 | 0,046266 |
| 33 | 0,077399 | 0,052189 |
| 34 | 0,082877 | 0,049732 |
| 35 | 0,079545 | 0,050883 |
| 36 | 0,083686 | 0,04983 |
| 37 | 0,078031 | 0,053976 |
| 38 | 0,090477 | 0,044704 |
| 39 | 0,080996 | 0,054017 |
| 40 | 0,083621 | 0,046886 |
| 41 | 0,082584 | 0,043804 |
| 42 | 0,085583 | 0,053386 |
| 43 | 0,088587 | 0,048049 |
| 44 | 0,088046 | 0,056176 |
| 45 | 0,08576 | 0,053923 |
| 46 | 0,093484 | 0,050588 |
| 47 | 0,08038 | 0,049077 |
| 48 | 0,089768 | 0,054062 |
| 49 | 0,088132 | 0,049468 |
| 50 | 0,092156 | 0,054198 |
| 51 | 0,091796 | 0,057601 |
| 52 | 0,094216 | 0,055128 |
| 53 | 0,087008 | 0,064363 |
| 54 | 0,100932 | 0,053505 |
| 55 | 0,091058 | 0,076117 |
| 56 | 0,090119 | 0,057502 |
| 57 | 0,099213 | 0,05532 |
| 58 | 0,094055 | 0,054585 |
| 59 | 0,094191 | 0,061915 |
| 60 | 0,094613 | 0,05996 |
| 61 | 0,100521 | 0,060198 |
| 62 | 0,094105 | 0,055895 |
| 63 | 0,096873 | 0,059207 |
| 64 | 0,097785 | 0,057035 |

Co obrazują poniższe wykresy:

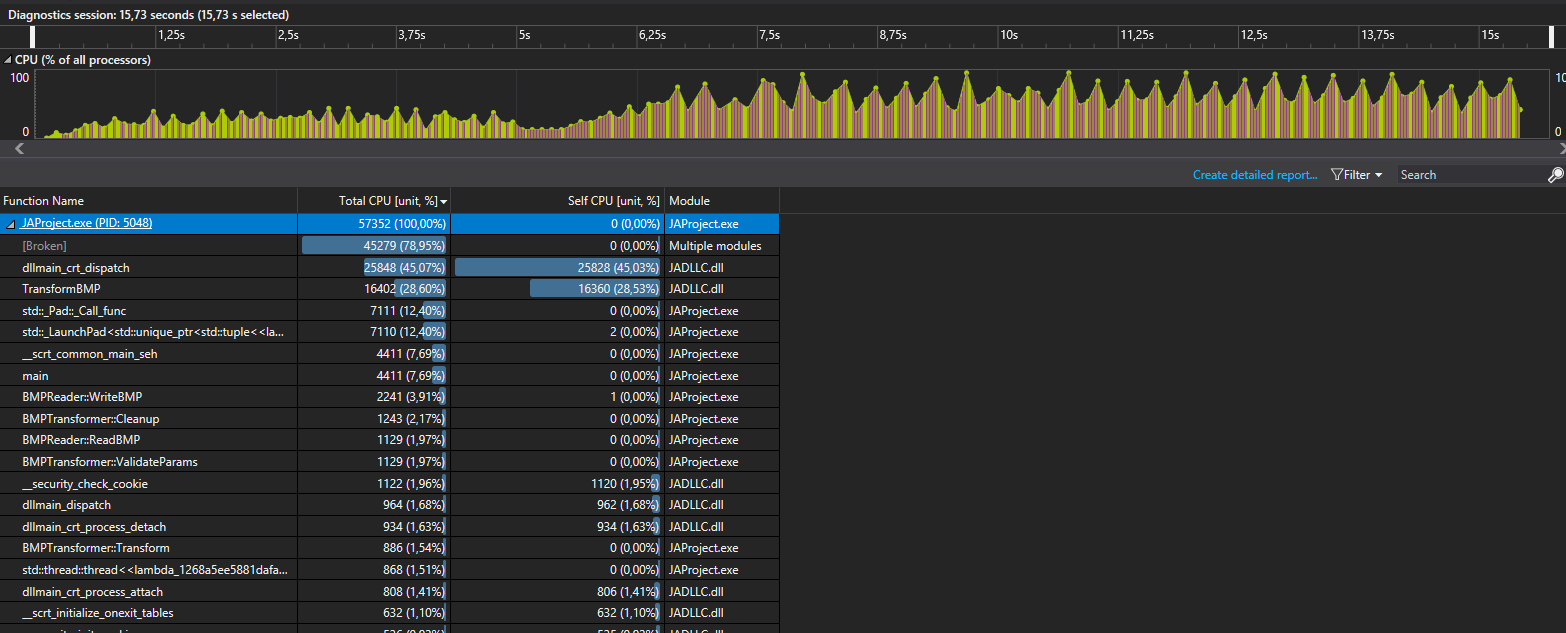
Zaobserwowano 2 krotne przyspieszenie programu kosztem około 1,5krotnie dłuższego kodu.

# Analiza działania programu z wykorzystaniem profiler’a VS2017.

Wykorzystując profiler 2017 dokonano pomiarów działania programu w trybie testowym.



Na podstawie niniejszego wykresu jesteśmy w stanie zauważyć widoczny wzrost w użyciu procesora w okolicy 5-tej sekundy. Zważając na wykresy czasów wykonania dllek asm i C/C++ oraz na kolejność testów (najpierw c++, następnie asm) wnioskujemy, że właśnie w tym momencie program zaczął wykonywać program w wersji asm.



Z powyższego wykresu możemy zobaczyć różnice w użyciu procesora dla poszczególnych funkcji. Widoczne jest, że funkcja

# Instrukcja obsługi programu JAproject.exe

Program jest aplikacją konsolową i nie posiada żadnego interfejsu graficznego. Wszelkie wywołania programu muszą zawierać w linii poleceń parametry, których znaczenie i sposób użycia został przedstawiony w sekcji opis parametrów programu.

Przykładowe wywołania:

Bez testów

-r 200 -g 100 -b 10 -A1C0 1 -t 1 -i /Image/bmpx.bmp -o /Image/bmpxout.bmp

Z testami

-act 0 -r 200 -g 100 -b 10 -A1C0 1 -t 1 -i /Image/bmpx.bmp -o /Image/bmpxout.bmp

Program wymaga obecności bibliotek JADLL.dll oraz JADLLC.dll w tym samym folderze co uruchamiany program. Program nie jest zabezpieczony przed błędnymi ścieżkami i błędnie umiejscowionymi plikami .dll.

## Wnioski

Podczas wykonywania tego projektu można było wysunąć wiele interesujących wniosków i nauczyć się pracy z asemblerem. Efektem ubocznym pracy nad wynikowym programem była również nauka współpracy z plikami bmp oraz konfigurowaniem środowiska pod biblioteki dynamiczne. To ostatnie, czyli linkowanie dynamiczne wydaje się być najbardziej przydatnym i istotnym z punktu widzenia zastosowania w przyszłości albowiem jest niezwykle wygodne i zdecydowanie przewyższa linkowanie statyczne pod względem wypuszczania aplikacji. Sam asembler i pisanie bibliotek w asemblerze, zdecydowanie warte docenienia, nie jest tak powszechne jak zastosowanie bibliotek z rozszerzeniem dll. Ponadto godnym zauważenia jest przewaga asemblera nad innymi językami pod względem przejrzystości wykonywanych operacji. Jednym z cenniejszych doświadczeń, które można było przeanalizować jest wyzbycie się na czas pracy w asemblerze pojęcia typu zmiennej. Podczas pracy z biblioteką napisaną w C/C++ pojawił się problem przepełniania wartości pikseli przez używanie wskaźnika na char zamiast na unsigned char. Ten błąd nie mógł się pojawić w bibliotece w asm, co sprawiło, że to właśnie ona poprawnie działała jako pierwsza i generowała optycznie ładny i płynny gradient.

# Literatura

<https://www.felixcloutier.com/x86/>

<https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf>

<https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/x64-architecture>

<https://software.intel.com/en-us/articles/introduction-to-x64-assembly>

<http://linasm.sourceforge.net/docs/instructions/index.php>

<https://solarianprogrammer.com/2018/11/19/cpp-reading-writing-bmp-images/>