Projekt z języków assemblerowych: Nakładanie gradientu na bitmapę

Autor: Tomasz Skowron
Inf sem 5

Założenia projektowe

- **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)**
- **4. W programie będzie można wybrać kolor do nałożenia.**
- **4. W programie dostępny będzie wybór ścieżek do plików we/wy**

Podstawowe klasy programu cz. 1

```
int CallMyDLL(unsigned char rg, unsigned char gg, unsigned char bg, int width, int height, void* data);
int CallMyDLLC(unsigned char rg, unsigned char gg, unsigned char bg, int width, int height, void* data);

struct BMPReader;
struct ParamReader;

struct BMPTransformer
{
    BMPReader* bmp;
    ParamReader* paramReader;
    char**argv;
    int argc;
    int*heightsarray;
    int height;
    int iterator;
    std::vector<std::thread*>threads;
public:
    BMPTransformer(char*_argv[], int _argc);
    ~BMPTransformer();
    void ReadFromFile();
    void ValidateParams();
    void CalculateHeights();
    void SaveBMP();
    void Transform();
    void Cleanup();
private:
```

```
struct ParamReader
{
    int NumberOfthreads=1;
    std::string infilepath;
    std::string outfilepath;
    bool A1C0switch = 1;
    void setParam(std::string param, std::string val);
    unsigned char red=200;
    unsigned char green=0;
    unsigned char blue=0;
    ParamReader();
};
```

Podstawowe klasy programu cz. 2

```
#define EIF ,true:false;
struct ParamReader;


////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)
```

Podstawowe klasy programu cz. 3

```cpp
#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)

class BMPReader
{
public:
    BMPFileHeader* bfh;
    BMPInfoHeader* bih;
    BMPColorHeader* bch;
    unsigned char*data;
    int width;
    int height;
    int r;
    int g;
    int b;
    BMPReader();
    ~BMPReader();
    int SetRgb(int R, int G, int B);
    int ReadBMP(std::string path,ParamReader* paramReader);
    int WriteBMP(std::string path);
    int BMPTransform(unsigned char r, unsigned char g, unsigned char b);
private:
    void* memory;
    int bmp_size;
};
```

Wykorzystane biblioteki

```
#include <string>
#include <fstream>
#include <iostream>
#include <Windows.h>
#include <ios>
#include <sstream>
#include <thread>
#include <vector>
#include <chrono>

#include "Utility.h"
#include "BMPReader.h"
#include "ParamReader.h"
#include "BMPTransformer.h"
```

Podlinkowanie bibliotek?

Podlinkowanie bibliotek? Dynamiczne

```
int CallMyDLL(unsigned char rg, unsigned char gg, unsigned char bg, int width, int height, void* data)
{
    HINSTANCE hGetProcIDDLL = LoadLibrary("JADLL.dll");
    FARPROC lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "TransformBMP");
    typedef int(__stdcall * pICFUNC)(unsigned char, unsigned char, unsigned char, int, int, void*);
    pICFUNC Algorithm;
    Algorithm = pICFUNC(lpfnGetProcessID);

    Algorithm(rg, gg, bg, width, height, data);

    FreeLibrary(hGetProcIDDLL);
    return 0;
}

int CallMyDLLC(unsigned char rg, unsigned char gg, unsigned char bg, int width, int height, void* data)
{
    HINSTANCE hGetProcIDDLL = LoadLibrary("JADLLC.dll");
    FARPROC lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "TransformBMP");
    typedef void(__cdecl * pICFUNC)(unsigned char, unsigned char, unsigned char, int, int, void*);
    pICFUNC Algorithm;
    Algorithm = pICFUNC(lpfnGetProcessID);

    Algorithm(rg, bg, gg, width, height, data);

    FreeLibrary(hGetProcIDDLL);
    return 0;
}
```

Algorytm- co to jest gradient?

Algorytm- co to jest gradient?

Aglorytm-omówienie na przykładzie implementacji w C/C++

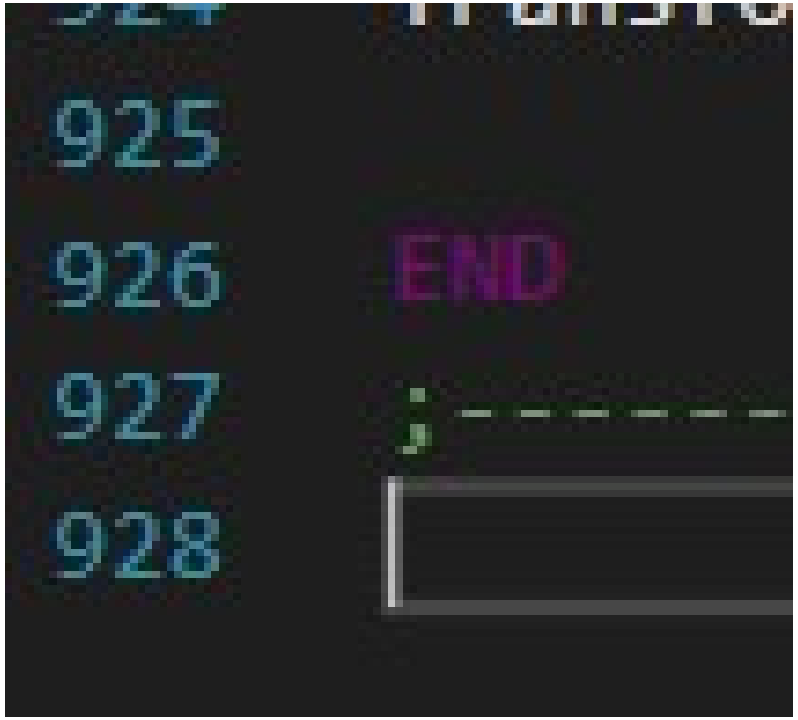
```
DLLEXPORT void TransformBMP(unsigned char RGradient, unsigned char GGradient, unsigned char BGradient, int width, int height, void* data)
{
    int remainder = (width * 3) % 4;
    unsigned char * temp = (unsigned char*)data;
    unsigned char CDR = RGradient;
    unsigned char CDG = GGradient;
    unsigned char CDB = BGradient;
    int divisor = 0;
    int max = (width)*height * 3;
    for (int i = 0; i < max; i += 3)
    {
        CDR = RGradient * (width - divisor) / width;
        CDG = GGradient * (width - divisor) / width;
        CDB = BGradient * (width - divisor) / width;

        divisor == width ?  divisor = 0 : 0;
        divisor += 1;
        temp[i] = (temp[i]+(CDG*(256 - temp[i])) / 256);
        temp[i + 1] = (temp[i+1]+(CDR*(256 - temp[i + 1])) / 256);
        temp[i + 2] = (temp[i + 2]+(CDB*(256 - temp[i + 2])) / 256);
        remainder ? i += remainder : 0;
    }
    return;
}
```

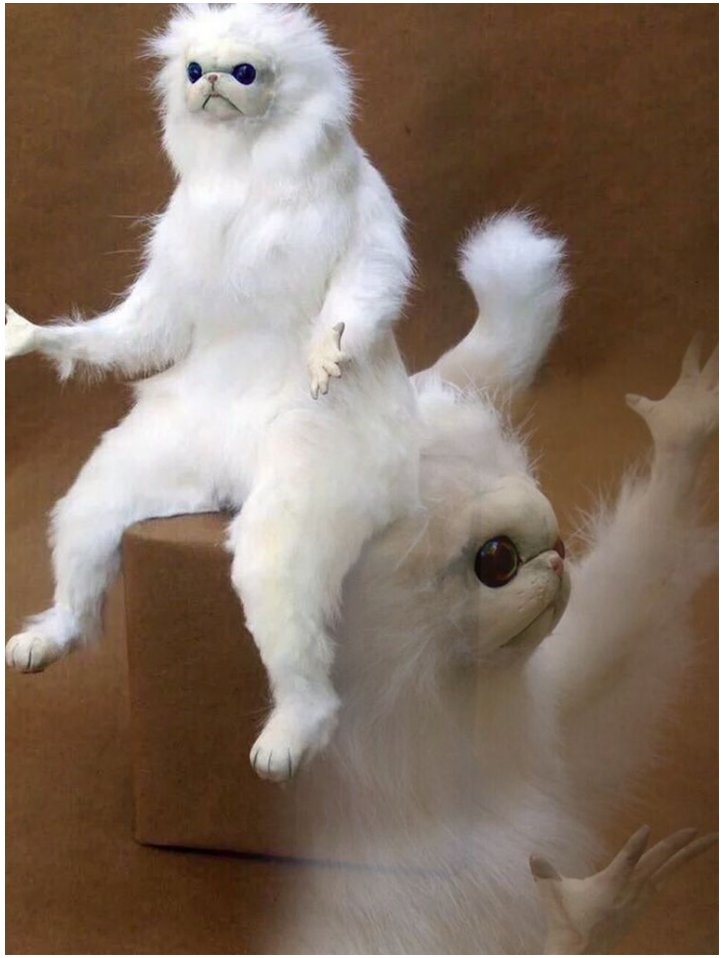
KAŻDY piksel zostanie uaktualniany o wartość zależną od jego osobistej POZYCJI i WARTOŚCI PRZED TRANSFOMRACJĄ!!!

Czy zatem algorytm nadaje się do implementacji w assemblerze?

A udało się?

A udało się?





Wykorzystanie instrukcji wektorowych

- movq
- movhlps
- movlhps
- xorps
- orps
- vzeroall
- movdqu (z nadzieją, że nadrobi pozostałe straty czasowe)

Idea schematu realizacji transformacji bitmapy w assemblerze cz.1

- Wyjęcie zmiennych z rejestrów, będą nam potrzebne
- 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}

Idea schematu realizacji transformacji bitmapy w assemblerze cz.2

- 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

– Sprawdzanie warunku końca danych

Algorytm testujący program
Boska klasa - BMPTransformer

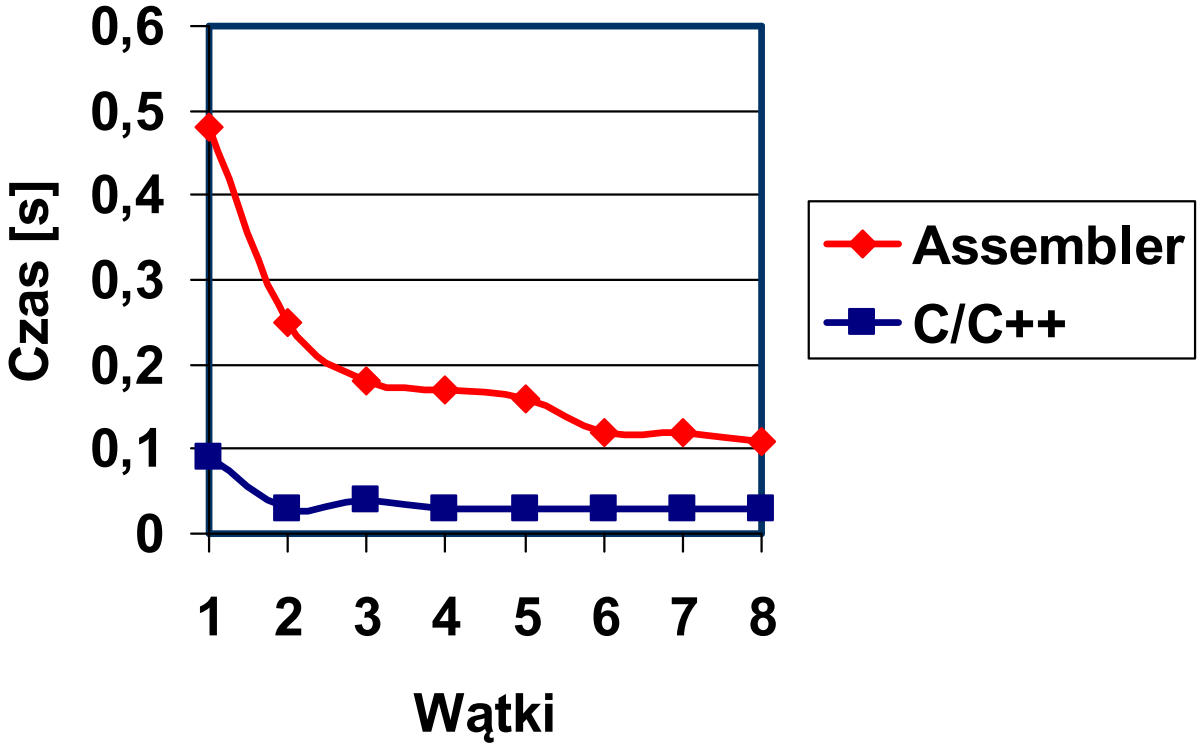
```
for (int i = 1; i <= std::thread::hardware_concurrency(); i++)
{
    BMPTransformer* bmpt = new BMPTransformer(argv, argc);
    bmpt->ReadFromFile();
    bmpt->ValidateParams();
    ///------------------------------
    //override params if necessary here
    bmpt->paramReader->A1C0switch = 0;
    bmpt->paramReader->NumberOfthreads = i;

    ///------------------------------
    bmpt->CalculateHeights();
    ///------------------------------
    //time testing area
    auto start = std::chrono::high_resolution_clock::now();

    bmpt->Transform();

    auto finish = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed = finish - start;
    double seconds = elapsed.count();
    DLLCstrs.push_back(std::string("Elapsed time: "+std::to_string(seconds)+"s with: "+std::to_string(i)+" threads\n"));

    ///------------------------------
    bmpt->SaveBMP();
    bmpt->Cleanup();
}
///--------------------------------
//send test results out to file
```

Co z wynikami?

Assembler
Elapsed time: 0.485349s with: 1 threads
Elapsed time: 0.258894s with: 2 threads
Elapsed time: 0.180423s with: 3 threads
Elapsed time: 0.177424s with: 4 threads
Elapsed time: 0.167070s with: 5 threads
Elapsed time: 0.128334s with: 6 threads
Elapsed time: 0.123208s with: 7 threads
Elapsed time: 0.113828s with: 8 threads

C/C++
Elapsed time: 0.099588s with: 1 threads
Elapsed time: 0.038282s with: 2 threads
Elapsed time: 0.040397s with: 3 threads
Elapsed time: 0.032899s with: 4 threads
Elapsed time: 0.028676s with: 5 threads
Elapsed time: 0.029681s with: 6 threads
Elapsed time: 0.027724s with: 7 threads
Elapsed time: 0.037474s with: 8 threads

Co z wynikami?



Wnioski

- Assembler jest fajny
- W rejestrze nie ma typu, są tylko zera i jedynki, a kiedy nie ma typów, to nie można użyć złego typu
- Dynamiczne linkowanie jest prostsze niż mogłoby się wydawać
- Wątki mają duży wpływ na szybkość działania programu, ale do czasu
- Nie każdy algorytm nadaje się do operacji wektorowych

Dziękuję za uwagę!