Университет ИТМО
Факультет программной инженерии и компьютерной техники

Низкоуровневое программирование

Лабораторная работа №8

Выполнила: Машина Е.А.

Группа P33113

Преподаватель: Логинов И.П.

Санкт-Петербург
2020 г.

Задание

In this assignment, we will create a program to perform a sepia filter on an image. A sepia filter makes an image with vivid colors look like an old, aged photograph. Most graphical editors include a sepia filter.

Код

main.c

```c
#include <stdio.h>
#include "picture.h"
#include "sepia.h"
#include <stdlib.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <unistd.h>

void write_ans(char* filename, image_t* img){
    switch (write_bmp(filename, img)){
        case WRITE_IMAGE_NOT_FOUND: {
            printf("Изображение для записи не найдено.\n");
            break;
        }
        case WRITE_FILENAME_NOT_FOUND: {
            printf("А тут как-то неправильно передается filename.\n");
            break;
        }
        case WRITE_ERROR: {
            printf("Незнаю, как такое получилось, но тут ошибка при открытии
файла.\n");
            break;
        }
        case WRITE_OK: {
            printf("Я записаль в %s :3\n", filename);
            break;
        }
        default: {
            printf("Совсем что-то не так пошло.\n");
            break;
        }
    }
}


int main() {
    image_t image_c;

    switch (read_bmp("s.bmp", &image_c)) {
        case READ_FILENAME_NOT_FOUND :{
            printf("Не найден файл.\n");
            return 1;
        }
        case READ_INVALID_BITS: {
            printf("Проблемы с данными.\n");
            return 1;
        }
        case READ_INVALID_HEADER: {
            printf("Проблемы с заголовком.\n");
            return 1;
        }
        case READ_OK:{
            printf("Изображение получено.\n");
            break;
```

```
        }
        default: {
            printf("Вообще что-то не так.\n");
            return 1;
        }
    }

    image_t image_asm;
    read_bmp("s.bmp", &image_asm);

    struct rusage r;
    struct timeval start;
    struct timeval end;
    getrusage(RUSAGE_SELF, &r);
    start = r.ru_utime;

    sepia_c_inplace(&image_c);

    getrusage(RUSAGE_SELF, &r);
    end = r.ru_utime;
    long res = ((end.tv_sec - start.tv_sec) * 1000000L) + end.tv_usec -
start.tv_usec;
    printf( "Время выполнения sepia (c): %ld\n", res);
    printf( "end.tv_sec: %ld\n", end.tv_sec);
    printf( "start.tv_sec: %ld\n", start.tv_sec);
    printf( "end.tv_usec: %ld\n", end.tv_usec);
    printf( "start.tv_usec: %ld\n", start.tv_usec);
    write_ans("out_c.bmp", &image_c);

    getrusage(RUSAGE_SELF, &r);
    start = r.ru_utime;
    sepia_sse_inplace(&image_asm);

    getrusage(RUSAGE_SELF, &r);
    end = r.ru_utime;
    res = ((end.tv_sec - start.tv_sec) * 1000000L) + end.tv_usec - start.tv_usec;
    printf( "Время выполнения sepia (sse): %ld\n", res);
    printf( "end.tv_sec: %ld\n", end.tv_sec);
    printf( "start.tv_sec: %ld\n", start.tv_sec);
    printf( "end.tv_usec: %ld\n", end.tv_usec);
    printf( "start.tv_usec: %ld\n", start.tv_usec);

    write_ans("out_sse.bmp", &image_asm);

    return 0;
}
```

picture.h

```
#ifndef LAB5_PICTURE_H
#define LAB5_PICTURE_H

#include <stdint.h>

typedef struct __attribute__((packed)){
    uint16_t bfType;
    uint32_t bfileSize;
    uint32_t bfReserved;
    uint32_t bOffBits;
    uint32_t biSize;
    uint32_t biWidth;
```

```
    uint32_t biHeight;
    uint16_t biPlanes;
    uint16_t biBitCount;
    uint32_t biCompression;
    uint32_t biSizeImage;
    uint32_t biXPelsPerMeter;
    uint32_t biYPelsPerMeter;
    uint32_t biClrUsed;
    uint32_t biClrImportant;
} bmp_header_t;

typedef struct {
    unsigned char b,g,r;
} pixel_t;

typedef struct {
    uint32_t width, height;
    pixel_t* data;
} image_t;

typedef enum {
    READ_OK = 0,
    READ_FILE_NOT_FOUND,
    READ_INVALID_BITS,
    READ_INVALID_HEADER,
    READ_FILENAME_NOT_FOUND
} read_error_code_t;

typedef enum {
    WRITE_OK = 0,
    WRITE_ERROR,
    WRITE_IMAGE_NOT_FOUND,
    WRITE_FILENAME_NOT_FOUND,
} write_error_code_t;

read_error_code_t read_bmp(char const* filename, image_t* new_image);

write_error_code_t write_bmp(char const* filename, image_t const* image);

#endif
```

picture.c

```
#include <stdlib.h>
#include <stdio.h>
#include "picture.h"

write_error_code_t write_bmp(char const *filename, image_t const *image) {

    if (image == NULL) {
        return WRITE_IMAGE_NOT_FOUND;
    }
    if (filename == NULL) {
        return WRITE_FILENAME_NOT_FOUND;
    }
    bmp_header_t *header = (bmp_header_t *) malloc(sizeof(bmp_header_t));
    int padding = image->width % 4;
    uint32_t i, j;
    image_t *new_image = (image_t *) malloc(sizeof(image_t));
    new_image->width = image->width + padding;
    new_image->height = image->height;
    new_image->data = (pixel_t *) calloc(1, new_image->height * new_image->width *
sizeof(pixel_t));
```

```
    for (i = 0; i < new_image->height; ++i) {
        for (j = 0; j < new_image->width; ++j) {
            if (j < new_image->width - padding) {
                *(new_image->data + i * new_image->width + j) = *(image->data + i *
image->width + j);
            }
        }
    }

    FILE *output = fopen(filename, "wb+");
    if (output == NULL) {
        return WRITE_ERROR;
    }
    header->bfType = 19778;
    header->bfileSize = new_image->width * new_image->height * sizeof(pixel_t) +
sizeof(header);
    header->bfReserved = 0;
    header->bOffBits = sizeof(bmp_header_t);
    header->biSize = 40;
    header->biPlanes = 0;
    header->biBitCount = 24;
    header->biCompression = 0;
    header->biSizeImage = new_image->width * new_image->height * sizeof(pixel_t);
    header->biXPelsPerMeter = 2835;
    header->biYPelsPerMeter = 2835;
    header->biClrUsed = 0;
    header->biClrImportant = 0;
    header->biWidth = new_image->width - padding;
    header->biHeight = new_image->height;

    fwrite(header, 1, sizeof(bmp_header_t), output);
    fwrite(image->data, 1, new_image->height * new_image->width * sizeof(pixel_t),
output);
    fclose(output);
    return WRITE_OK;
}

read_error_code_t read_bmp(char const *filename, image_t *input_image) {

    if (filename == NULL) {
        return READ_FILENAME_NOT_FOUND;
    }
    FILE *input = fopen(filename, "rb");
    if (input == NULL) {
        return READ_FILE_NOT_FOUND;
    }
    bmp_header_t header;
    fread(&header, 1, sizeof(header), input);
    if (header.bfType == 0) {
        return READ_INVALID_HEADER;
    }
    uint8_t *data = (uint8_t *) malloc(header.biSizeImage);
    fseek(input, header.bOffBits, SEEK_SET);
    fread(data, 1, header.biSizeImage, input);
    if (data == NULL) {
        return READ_INVALID_BITS;
    }
    if (input_image == NULL) {
        input_image = (image_t *) malloc(sizeof(image_t));
    }
```

```
    input_image->data = (pixel_t *) malloc(header.biHeight * header.biWidth *
sizeof(pixel_t));

    int padding = header.biWidth % 4;
    for (uint32_t i = 0; i < header.biHeight; ++i) {
        for (uint32_t j = 0; j < header.biWidth; ++j) {
            *(input_image->data + i * header.biWidth + j) = *(pixel_t *) (((uint8_t
*) data) +

sizeof(pixel_t) * (i * header.biWidth + j) +  padding * i);
        }
    }
    input_image->height = header.biHeight;
    input_image->width = header.biWidth;
    fclose(input);
    return READ_OK;
}
```

sepia.h

```
#ifndef LAB7_SEPIA_H
#define LAB7_SEPIA_H

#include "picture.h"

void sepia_c_inplace(image_t *img);

void sepia_sse_inplace(image_t *img);

#endif
```

sepia.c

```
#include "sepia.h"
#include <inttypes.h>
#include <glob.h>
#include <stdio.h>

static unsigned char sat( uint64_t x) {
    if (x < 256)
        return (unsigned char)x;
    return 255;
}

static void sepia_one(pixel_t* const pixel ) {
    static const float c[3][3] =  {
            { .393f, .769f, .189f },
            { .349f, .686f, .168f },
            { .272f, .543f, .131f } };
    pixel_t const old = *pixel;
    pixel->r = sat( (uint64_t) (old.r * c[0][0] + old.g * c[0][1] + old.b *
c[0][2]));
    pixel->g = sat( (uint64_t)(old.r * c[1][0] + old.g * c[1][1] + old.b  *
c[1][2]));
    pixel->b = sat( (uint64_t) (old.r * c[2][0] + old.g * c[2][1] + old.b *
c[2][2]));
}

void sepia_c_inplace(image_t* pic) {
    for (size_t i = 0; i < pic->height; i++) {
        for (size_t j = 0; j < pic->width; j++) {
            sepia_one(pic->data + i*pic->width + j);
        }
    }
```

```
}

void image_sepia_sse(pixel_t* pixel, uint32_t size);

void sepia_sse_inplace(image_t *img) {
    if (img->height * img->width < 4){
        for (int i = 0; i < img->height * img->width; ++i){
            sepia_one(img->data + i);
        }
        return;
    }
    image_sepia_sse(img->data, img->height * img->width - (img->height * img-
>width) % 4);
    for (int i  = img->height * img->width - (img->height * img->width) % 4; i <
img->height * img->width; ++i){
        sepia_one(img->data + i);
    }
}
```

image_sepia_sse.asm

```
global image_sepia_sse

section .data

; матрица для rgbr
align 16
c1_rgbr: dd 0.393, 0.349, 0.272, 0.393
align 16
c2_rgbr: dd 0.769, 0.686, 0.543, 0.769
align 16
c3_rgbr: dd 0.189, 0.168, 0.131, 0.189

; матрица для gbrg
align 16
c1_gbrg: dd 0.349, 0.272, 0.393, 0.349
align 16
c2_gbrg: dd 0.686, 0.543, 0.769, 0.686
align 16
c3_gbrg: dd 0.168, 0.131, 0.189, 0.168

; матрица для brgb
align 16
c1_brgb: dd 0.272, 0.393, 0.349, 0.272
align 16
c2_brgb: dd 0.543, 0.769, 0.686, 0.543
align 16
c3_brgb: dd 0.131, 0.189, 0.168, 0.131

align 16
shuffle_rgb_to_bgr: db 2, 1, 0, 5, 4, 3, 8, 7, 6, 11, 10, 9, -1, -1, -1, -1

section .text

; rdi = указатель на массив пикселей
; rsi = количество пикселей

%define xmm_ch1 xmm0
%define xmm_ch2 xmm1
%define xmm_ch3 xmm2

%define xmm_c1_rgbr xmm3
```

```
%define xmm_c2_rgbr xmm4
%define xmm_c3_rgbr xmm5

%define xmm_c1_gbrg xmm6
%define xmm_c2_gbrg xmm7
%define xmm_c3_gbrg xmm8

%define xmm_c1_brgb xmm9
%define xmm_c2_brgb xmm10
%define xmm_c3_brgb xmm11

%define xmm_rgbr xmm12
%define xmm_gbrg xmm13
%define xmm_brgb xmm14

%define xmm_shuffle_rgb_to_bgr xmm15

%define pixel_ptr r8

image_sepia_sse:
  movaps xmm_c1_rgbr, [c1_rgbr]
  movaps xmm_c2_rgbr, [c2_rgbr]
  movaps xmm_c3_rgbr, [c3_rgbr]

  movaps xmm_c1_gbrg, [c1_gbrg]
  movaps xmm_c2_gbrg, [c2_gbrg]
  movaps xmm_c3_gbrg, [c3_gbrg]

  movaps xmm_c1_brgb, [c1_brgb]
  movaps xmm_c2_brgb, [c2_brgb]
  movaps xmm_c3_brgb, [c3_brgb]

  movdqa xmm_shuffle_rgb_to_bgr, [shuffle_rgb_to_bgr]

  mov pixel_ptr, rdi
  lea rsi, [rsi + 2*rsi]
  add rsi, rdi

image_sepia_sse_loop_4_pixels:

  mov rdx, [pixel_ptr]

  movd xmm_ch3, edx
  pmovzxbd xmm_ch3, xmm_ch3
  shufps xmm_ch3, xmm_ch3, 0b11000000
  cvtdq2ps xmm_ch3, xmm_ch3

  shr rdx, 8
  movd xmm_ch2, edx
  pmovzxbd xmm_ch2, xmm_ch2
  shufps xmm_ch2, xmm_ch2, 0b11000000
  cvtdq2ps xmm_ch2, xmm_ch2

  shr rdx, 8
  movd xmm_ch1, edx
  pmovzxbd xmm_ch1, xmm_ch1
  shufps xmm_ch1, xmm_ch1, 0b11000000
  cvtdq2ps xmm_ch1, xmm_ch1

  xorps xmm_rgbr, xmm_rgbr
  vfmadd231ps xmm_rgbr, xmm_ch1, xmm_c1_rgbr
  vfmadd231ps xmm_rgbr, xmm_ch2, xmm_c2_rgbr
```

```
vfmadd231ps xmm_rgbr, xmm_ch3, xmm_c3_rgbr


shr rdx, 8
movd xmm_ch3, edx
pmovzxbd xmm_ch3, xmm_ch3
shufps xmm_ch3, xmm_ch3, 0b11110000
cvtdq2ps xmm_ch3, xmm_ch3

mov rdx, [pixel_ptr + 4]
movd xmm_ch2, edx
pmovzxbd xmm_ch2, xmm_ch2
shufps xmm_ch2, xmm_ch2, 0b11110000
cvtdq2ps xmm_ch2, xmm_ch2

shr rdx, 8
movd xmm_ch1, edx
pmovzxbd xmm_ch1, xmm_ch1
shufps xmm_ch1, xmm_ch1, 0b11110000
cvtdq2ps xmm_ch1, xmm_ch1

xorps xmm_gbrg, xmm_gbrg
vfmadd231ps xmm_gbrg, xmm_ch1, xmm_c1_gbrg
vfmadd231ps xmm_gbrg, xmm_ch2, xmm_c2_gbrg
vfmadd231ps xmm_gbrg, xmm_ch3, xmm_c3_gbrg


shr rdx, 8
movd xmm_ch3, edx
pmovzxbd xmm_ch3, xmm_ch3
shufps xmm_ch3, xmm_ch3, 0b11111100
cvtdq2ps xmm_ch3, xmm_ch3

shr rdx, 8
movd xmm_ch2, edx
pmovzxbd xmm_ch2, xmm_ch2
shufps xmm_ch2, xmm_ch2, 0b11111100
cvtdq2ps xmm_ch2, xmm_ch2

shr rdx, 8
movd xmm_ch1, edx
pmovzxbd xmm_ch1, xmm_ch1
shufps xmm_ch1, xmm_ch1, 0b11111100
cvtdq2ps xmm_ch1, xmm_ch1

xorps xmm_brgb, xmm_brgb
vfmadd231ps xmm_brgb, xmm_ch1, xmm_c1_brgb
vfmadd231ps xmm_brgb, xmm_ch2, xmm_c2_brgb
vfmadd231ps xmm_brgb, xmm_ch3, xmm_c3_brgb


cvtps2dq xmm_rgbr, xmm_rgbr
cvtps2dq xmm_gbrg, xmm_gbrg
cvtps2dq xmm_brgb, xmm_brgb


packssdw xmm_rgbr, xmm_gbrg

packssdw xmm_brgb, xmm_brgb

packuswb xmm_rgbr, xmm_brgb
```

```
pshufb xmm_rgbr, xmm_shuffle_rgb_to_bgr

pextrq rdx, xmm_rgbr, 0
mov [pixel_ptr], rdx
pextrd edx, xmm_rgbr, 2
mov [pixel_ptr + 8], edx


lea pixel_ptr, [pixel_ptr + 12]
cmp pixel_ptr, rsi
jl image_sepia_sse_loop_4_pixels

ret
```

Вывод

Выполнив эту лабораторную работу, я реализовала фильтр sepia для bmp изображения.