Федеральное государственное бюджетное образовательное учреждение
высшего образования

«Сибирский государственный университет телекоммуникаций и
информатики»

(СибГУТИ)

Кафедра прикладной математики и кибернетики

Лабораторная работа №4
по курсу программирование графических процессов

Выполнили:
студенты группы ИП-014

Гулая А.С.
Обухов А.И.
Малышев В.А.

Работу проверил:
доцент каф. ПМиК
Перцев И.В.

Новосибирск 2024 г.

Задание:

Вывести на экран 16, 256-цветный и True Color BMP файл используя putpixel, без использования библиотек обработки графических файлов.

Листинг программы:

```
#include <SDL2/SDL.h>
#include <SDL2/SDL_opengl.h> // otherwise we want to use OpenGL
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#pragma pack(1)
typedef struct {
    uint16_t signature;
    uint32_t filesize;
    uint32_t reserved;
    uint32_t offset;
    uint32_t header_size;
    uint32_t width;
    uint32_t height;
    uint16_t planes;
    uint16_t bpp;
    uint32_t compression;
    uint32_t image_size;
    uint32_t x_pixels_per_m;
    uint32_t y_pixels_per_m;
    uint32_t colors_used;
    uint32_t colors_important;
} Head;

typedef struct {
    Head head;
    char* filename;
    uint8_t* rastr;
    uint8_t* palette;
} Image;

void read_head(Head* head, FILE* file) { fread(head, sizeof(Head), 1,
file); }

Image read_image(FILE* file)
{
    Image image = { 0 };

    read_head(&image.head, file);

    fseek(file, sizeof(image.head), SEEK_SET);

    if (image.head.bpp <= 8) {
        int colors_number = 1 << image.head.bpp;
        image.palette = malloc(colors_number * 4 * sizeof(uint8_t));
        fread(image.palette, sizeof(uint8_t), (colors_number * 4),
file);
```

```
    }

    image.rastr = malloc(image.head.filesize);
    int bytes_per_pixel = image.head.bpp / 8;
    if (bytes_per_pixel == 0) {
        bytes_per_pixel = 1;
    }
    int row_size = image.head.width * bytes_per_pixel;

    uint32_t padding = (4 - image.head.width * bytes_per_pixel % 4) %
4;
    for (int i = 0; i < image.head.height; i++) {
        fread(&image.rastr[i * row_size], sizeof(uint8_t), row_size,
file);
        fseek(file, padding, SEEK_CUR);
    }

    fread(image.rastr, sizeof(uint8_t), image.head.filesize, file);

    return image;
}

void print_head(Head head)
{
    printf("signature: %s\n", (char*)&head.signature);
    printf("filesize: %d\n", head.filesize);
    printf("reserved: %d\n", head.reserved);
    printf("offset: %d\n", head.offset);
    printf("header_size: %d\n", head.header_size);
    printf("width: %d\n", head.width);
    printf("height: %d\n", head.height);
    printf("planes: %d\n", head.planes);
    printf("bpp: %d\n", head.bpp);
    printf("compression: %d\n", head.compression);
    printf("image_size: %d\n", head.image_size);
    printf("x_pixels_per_m: %d\n", head.x_pixels_per_m);
    printf("y_pixels_per_m: %d\n", head.y_pixels_per_m);
    printf("colors_used: %d\n", head.colors_used);
    printf("colors_important: %d\n", head.colors_important);
}

void process_true_color(Image image, SDL_Renderer* renderer)
{
    for (int i = 0; i < image.head.height; i++) {
        for (int j = 0; j < image.head.width; j++) {
            uint8_t bytes_per_pixel = image.head.bpp / 8;
            uint64_t actual_buffer_size = image.head.width *
image.head.height * bytes_per_pixel;
            uint32_t offset = actual_buffer_size - (i *
image.head.width * bytes_per_pixel + j * bytes_per_pixel);

            uint8_t r = image.rastr[offset + 2];
            uint8_t g = image.rastr[offset + 1];
            uint8_t b = image.rastr[offset];

            SDL_SetRenderDrawColor(renderer, r, g, b, 0xFF);
            SDL_RenderDrawPoint(renderer, j, i);
        }
```

```
    }
}

void process_8_bit(Image image, SDL_Renderer* renderer)
{
    for (int i = 0; i < image.head.height; i++) {
        for (int j = 0; j < image.head.width; j++) {
            uint8_t palette_index = image.rastr[(image.head.height - 1
- i) * image.head.width + j];
            uint32_t palette_offset = palette_index * 4;

            uint8_t r = image.palette[palette_offset + 2];
            uint8_t g = image.palette[palette_offset + 1];
            uint8_t b = image.palette[palette_offset];

            SDL_SetRenderDrawColor(renderer, r, g, b, 0xFF);
            SDL_RenderDrawPoint(renderer, j, i);
        }
    }
}

void process_4_bit(Image image, SDL_Renderer* renderer)
{
    for (int i = 0; i < image.head.height; i++) {
        for (int j = 0; j < image.head.width; j++) {
            uint8_t palette_index = image.rastr[(image.head.height - 1
- i) * ((image.head.width + 1) / 2) + j / 2];
            uint8_t palette_offset;

            if (j % 2 == 0) {
                palette_offset = ((palette_index) >> 4) * 4;
            } else {
                palette_offset = (palette_index & 0x0F) * 4;
            }

            uint8_t r = image.palette[palette_offset + 2];
            uint8_t g = image.palette[palette_offset + 1];
            uint8_t b = image.palette[palette_offset];
            SDL_SetRenderDrawColor(renderer, r, g, b, 0xFF);
            SDL_RenderDrawPoint(renderer, j, i);
        }
    }
}

void display_bmp(Image image)
{
    SDL_Init(SDL_INIT_VIDEO);
    SDL_Window* window = SDL_CreateWindow(
        image.filename, SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
        image.head.width, image.head.height, SDL_WINDOW_OPENGL);

    if (!window) {
        fprintf(stderr, "Error failed to create window!\n");
        exit(1);
    }

    SDL_GLContext context = SDL_GL_CreateContext(window);
```

```
    if (!context) {
        fprintf(stderr, "Error failed to create a context\n!");
        exit(1);
    }

    SDL_Event event;
    bool running = true;

    SDL_Renderer* renderer = SDL_CreateRenderer(window, -1,
SDL_RENDERER_ACCELERATED);

    if (NULL != image.palette) {
        if (image.head.bpp == 8) {
            process_8_bit(image, renderer);
        } else {
            process_4_bit(image, renderer);
        }
    } else {
        process_true_color(image, renderer);
    }

    SDL_SetRenderDrawColor(renderer, 0xFF, 0x00, 0x00, 0xFF);
    SDL_RenderPresent(renderer);

    while (running) {
        while (SDL_PollEvent(&event)) {
            running = event.type != SDL_QUIT;
        }
    }

    SDL_GL_DeleteContext(context);

    SDL_DestroyWindow(window);

    SDL_Quit();
}

int main(int argc, char* argv[])
{
    if (argc <= 1) {
        fprintf(stderr, "Usage: %s <input>\n", argv[0]);
        return -1;
    }

    char* filename = argv[1];

    FILE* file = fopen(filename, "rb");
    if (NULL == file) {
        perror(filename);
        return -1;
    }

    Image image = read_image(file);
    image.filename = filename;

    print_head(image.head);

    display_bmp(image);
```

```
    fclose(file);
    return 0;
}
```

Результат работы:

Рис. 1 16-цветный BMP файл

Рис. 2 256-цветный BMP файл

Рис. 3 True Color BMP файл