Университет ИТМО

Кафедра информатики и прикладной математики

Лабораторная работа №2

по дисциплине   
«Проектирование человеко-машинного интерфейса»

Вариант 16

Выполнил:

Райла Мартин,

студент группы Р3311.

Преподаватель:

Зинчик Александр Адольфович

Санкт-Петербург – 2017

**Задание:**

Написать программу, выполняющую следующее задание. Для произвольного графического файла формата BMP размером не менее чем 2000х2000 выполнить следующие действия:

1)Загрузить и отобразить файл.

2) Применить действие, согласно варианту задания.   
**«Изменить цветовые каналы изображения по схеме B->G->R->B»**Пользоваться функциями GetPixel и SetPixel запрещено.

3)Отобразить результат и сохранить новый файл.

**Часть 1.Код программы:**

/\*\*

Source.cpp

Purpose: Changes bmp file

**@author** Rayla Martin

**@version** 1.0 10.11.2017

\*/

#include <windows.h>

#include <time.h>

#include <stdio.h>

#include <iostream>

#include <fstream>

#include "resource.h"

#include <Commdlg.h>

#include <winuser.h>

/// This method is used as application entry point

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)

{

HWND hWnd;

WNDCLASS WndClass;

MSG Msg;

wchar\_t szClassName[] = L"Лабораторная работа №2";

// Set window attributes

WndClass.style = CS\_HREDRAW | CS\_VREDRAW;

WndClass.lpfnWndProc = handleWindowEvents;

WndClass.cbClsExtra = 0;

WndClass.cbWndExtra = 0;

WndClass.hInstance = hInstance;

WndClass.hIcon = LoadIcon(NULL, IDI\_WINLOGO);

WndClass.hCursor = LoadCursor(NULL, IDC\_ARROW);

WndClass.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

WndClass.lpszMenuName = L"Menu";

WndClass.lpszClassName = szClassName;

// Register window class

if (!RegisterClass(&WndClass))

{

MessageBox(NULL, L"Cannot register class", L"Error", MB\_OK);

return 0;

}

// Create window

hWnd = CreateWindow(szClassName, L"Лабораторная работа №2",

WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT,

CW\_USEDEFAULT, CW\_USEDEFAULT,

CW\_USEDEFAULT, NULL, NULL,

hInstance, NULL);

HMENU MainMenu = CreateMenu();

AppendMenu(MainMenu, MF\_STRING, OPEN\_ID, L"Open");

AppendMenu(MainMenu, MF\_STRING, RUN\_MAINTASK\_ID, L"Apply");

AppendMenu(MainMenu, MF\_STRING, IDM\_ABOUT, L"About");

SetMenu(hWnd, MainMenu);

if (!hWnd)

{

MessageBox(NULL, L"Cannot create window", L"Error", MB\_OK);

return 0;

}

// Show window, start handling his messages

ShowWindow(hWnd, SW\_SHOWDEFAULT);

UpdateWindow(hWnd);

while (GetMessage(&Msg, NULL, 0, 0))

{

TranslateMessage(&Msg);

DispatchMessage(&Msg);

}

return Msg.wParam;

}

/// This method is used for handle messages in the queue

LRESULT CALLBACK handleWindowEvents(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)

{

static HINSTANCE hInstance;

switch (message)

{

case WM\_CREATE:

{

hInstance = ((LPCREATESTRUCT)lParam)->hInstance;

return 0;

}

case WM\_DESTROY:

{

PostQuitMessage(0);

break;

}

case WM\_PAINT:

{

BITMAP bm;

PAINTSTRUCT ps;

HDC hDC = BeginPaint(hwnd, &ps);

HDC hCompatibleDC = CreateCompatibleDC(hDC);

HBITMAP hOldBitmap = (HBITMAP)SelectObject(hCompatibleDC, hBitmap);

GetObject(hBitmap, sizeof(bm), &bm);

BitBlt(hDC, 0, 0, bm.bmWidth, bm.bmHeight, hCompatibleDC, 0, 0, SRCCOPY);

SelectObject(hCompatibleDC, hOldBitmap);

DeleteDC(hCompatibleDC);

EndPaint(hwnd, &ps);

break;

}

case WM\_COMMAND:

{

if (LOWORD(wParam) == IDM\_ABOUT)

{

DialogBox(hInstance, L"AboutBox", hwnd, AboutDlgProc);

return 0;

}

if (LOWORD(wParam) == OPEN\_ID)

{

LPCTSTR fileName = getFileName();

hBitmap = (HBITMAP)LoadImage(0, fileName, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE | LR\_CREATEDIBSECTION);

InvalidateRect(hwnd, NULL, TRUE);

if\_is\_loaded = true;

}

if (LOWORD(wParam) == RUN\_MAINTASK\_ID)

{

if (!if\_is\_loaded)

{

break;

}

runReplaceColorTask(hwnd, hBitmap);

}

break;

}

default:

{

return DefWindowProc(hwnd, message, wParam, lParam);

}

}

return 0;

}

/// This method is used for display information about author.

BOOL CALLBACK AboutDlgProc(HWND hDlg, UINT iMsg, WPARAM wParam, LPARAM lParam)

{

switch (iMsg)

{

case WM\_INITDIALOG:

return TRUE;

case WM\_PAINT:

{

BITMAP bm;

PAINTSTRUCT ps;

HDC hDC = BeginPaint(hDlg, &ps);

HDC hCompatibleDC = CreateCompatibleDC(hDC);

HBITMAP hOldBitmap = (HBITMAP)SelectObject(hCompatibleDC, hBitmap);

GetObject(hBitmap, sizeof(bm), &bm);

BitBlt(hDC, 0, 0, bm.bmWidth, bm.bmHeight, hCompatibleDC, 0, 0, SRCCOPY);

SelectObject(hCompatibleDC, hOldBitmap);

DeleteDC(hCompatibleDC);

EndPaint(hDlg, &ps);

break;

}

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case IDOK:

case IDCANCEL:

EndDialog(hDlg, 0);

return TRUE;

}

break;

}

return FALSE;

}

void runReplaceColorTask(HWND hWnd, HBITMAP hBitmap)

{

BITMAP bitmap;

PAINTSTRUCT paintstruct;

static bmpSize\_t bmpRenderSize;

bmpRenderSize.width =

bmpRenderSize.height = 400;

auto deviceContext = BeginPaint(hWnd, &paintstruct);

auto tmpContext = CreateCompatibleDC(deviceContext);

auto oldBitmap = SelectObject(tmpContext, hBitmap);

GetObject(hBitmap, sizeof(bitmap), &bitmap);

StretchBlt(

deviceContext, 0, 0, bmpRenderSize.width, bmpRenderSize.height,

tmpContext, 0, 0, bitmap.bmWidth, bitmap.bmHeight,

SRCCOPY

);

changeBMPColors(tmpContext, hBitmap, bitmap);

StretchBlt(

deviceContext, bmpRenderSize.width, 0, bmpRenderSize.width,

bmpRenderSize.height,

tmpContext, 0, 0, bitmap.bmWidth, bitmap.bmHeight,

SRCCOPY

);

HBITMAP imageToSave = (HBITMAP)SelectObject(tmpContext, oldBitmap);

saveBitmap(tmpContext, imageToSave, bitmap.bmWidth, bitmap.bmHeight);

InvalidateRect(hWnd, NULL, TRUE);

DeleteDC(tmpContext);

EndPaint(hWnd, &paintstruct);

}

static void changeBMPColors(HDC deviceCtx, HBITMAP bmpImage, BITMAP bmpInfo)

{

char const sizeOfPixels = sizeof(int32\_t);

BITMAPINFO bi;

auto imgHeight = bmpInfo.bmHeight;

auto imgWidth = bmpInfo.bmWidth;

char\* buf = new char[imgHeight \* imgWidth \* sizeOfPixels];

bi.bmiHeader.biSize = sizeof(bi.bmiHeader);

bi.bmiHeader.biWidth = imgWidth;

bi.bmiHeader.biHeight = imgHeight;

bi.bmiHeader.biPlanes = 1;

bi.bmiHeader.biBitCount = sizeOfPixels \* 8;

bi.bmiHeader.biCompression = BI\_RGB;

bi.bmiHeader.biSizeImage = imgWidth \* sizeOfPixels \* imgHeight;

bi.bmiHeader.biClrUsed = 0;

bi.bmiHeader.biClrImportant = 0;

// retrieves the bits of the specified compatible bitmap and copies them into a buffer as a DIB using the specified format.

// bRes - is the number of scan lines copied from the bitmap.

int bRes = GetDIBits(

deviceCtx, bmpImage, 0, bmpInfo.bmHeight, buf, &bi, DIB\_RGB\_COLORS

);

long bufSize = imgHeight \* imgWidth \* sizeOfPixels;

for (auto i = 0; i < bufSize; i += sizeOfPixels)

{

auto blue = buf[i];

auto green = buf[i + 1];

auto red = buf[i + 2];

buf[i] = red; // blue - red; R->B

buf[i + 1] = blue; // green - blue; B->G

buf[i + 2] = green; // red - green; G->R

}

SetDIBits(deviceCtx, bmpImage, 0, bRes, buf, &bi, DIB\_RGB\_COLORS);

delete[] buf;

}

/// This method just return a file name.

LPCTSTR getFileName()

{

OPENFILENAME ofn;

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = NULL;

ofn.lpstrFile = new WCHAR[1000];

ofn.lpstrFile[0] = '\0';

ofn.nMaxFile = 1000;

ofn.lpstrFilter = L"BMP\0\*.bmp\0";

ofn.nFilterIndex = 1;

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = NULL;

ofn.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

GetOpenFileName(&ofn);

return ofn.lpstrFile;

}

/// This method just save new bmp file.

int saveBitmap(HDC hdc, HBITMAP H, int width, int height)

{

BITMAPFILEHEADER bmfHeader;

BITMAPINFOHEADER bi;

bi.biSize = sizeof(BITMAPINFOHEADER);

bi.biWidth = width;

bi.biHeight = height;

bi.biPlanes = 1;

bi.biBitCount = 32;

bi.biCompression = BI\_RGB;

bi.biSizeImage = 0;

bi.biXPelsPerMeter = 0;

bi.biYPelsPerMeter = 0;

bi.biClrUsed = 0;

bi.biClrImportant = 0;

DWORD dwBmpSize = ((width \* bi.biBitCount + 31) / 32) \* 4 \* height;

HANDLE hDIB = GlobalAlloc(GHND, dwBmpSize);

char \*lpbitmap = (char \*)GlobalLock(hDIB);

GetDIBits(hdc, H, 0,

(UINT)height,

lpbitmap,

(BITMAPINFO \*)&bi, DIB\_RGB\_COLORS);

HANDLE hFile = CreateFile(L"C:/Users/admin/Desktop/tphoto/result.bmp",

GENERIC\_WRITE,

0,

NULL,

CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL, NULL);

DWORD dwSizeofDIB = dwBmpSize + sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER);

bmfHeader.bfOffBits = (DWORD)sizeof(BITMAPFILEHEADER) + (DWORD)sizeof(BITMAPINFOHEADER);

bmfHeader.bfSize = dwSizeofDIB;

bmfHeader.bfType = 0x4D42; //BM

DWORD dwBytesWritten = 0;

WriteFile(hFile, (LPSTR)&bmfHeader, sizeof(BITMAPFILEHEADER), &dwBytesWritten, NULL);

WriteFile(hFile, (LPSTR)&bi, sizeof(BITMAPINFOHEADER), &dwBytesWritten, NULL);

WriteFile(hFile, (LPSTR)lpbitmap, dwBmpSize, &dwBytesWritten, NULL);

GlobalUnlock(hDIB);

GlobalFree(hDIB);

CloseHandle(hFile);

return 0;

}