Университет ИТМО

Кафедра информатики и прикладной математики

Лабораторная работа №1

по дисциплине   
«Проектирование человеко-машинного интерфейса»

Вариант 16

Выполнил:

Райла Мартин,

студент группы Р3311.

Преподаватель:

Зинчик Александр Адольфович

Санкт-Петербург – 2017

**Задание:**

Изучить работу функции BitBlt. Выполнить практическое задание.

1. 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:   
**Увеличить интенсивность синего и красного канала изображения до максимума**.   
Пользоваться функциями GetPixel и SetPixel запрещено.

3)Отобразить результат и сохранить новый файл.

2. Оценить отношение скорости работы функции BitBlt и пары функций GetPixel и SetPixel при решении вашего варианта задания и построить график зависимости этого отношения от размеров изображения. Размеры изображения от 200х200 до 2000х2000 с шагом увеличения каждого размера на 200 (200х200, 400х400 и т.д.)

**Часть 1.Код программы:**

#include <windows.h>

#include <time.h>

#include <stdio.h>

#include <iostream>

#include <fstream>

#include "resource.h"

#include <Commdlg.h>

#include <io.h>

#include <fcntl.h>

#include <winuser.h>

#include <debugapi.h>

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

{

HWND hWnd;

WNDCLASS WndClass;

MSG Msg;

wchar\_t szClassName[] = L"Лабораторная работа №1";

// 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 №1",

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\_BITBLT\_ID, L"Change");

AppendMenu(MainMenu, MF\_STRING, RUN\_SETPIXEL\_ID, L"SetPixel");

AppendMenu(MainMenu, MF\_STRING, RUN\_TESTS\_ID, L"Tests");

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;

}

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);

isLoaded = true;

}

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

{

if (!isLoaded)

{

break;

}

if (runBitBltFilter(hwnd, hBitmap) == 1) // if error

{

break;

}

}

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

{

if (!isLoaded)

{

break;

}

runSetPixelFilter(hwnd, hBitmap);

}

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

{

runTests(hwnd);

}

break;

}

default:

{

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

}

}

return 0;

}

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;

}

int runBitBltFilter(HWND hWnd, HBITMAP hBitmap)

{

BITMAP bitmap;

HDC hCompatibleDC = CreateCompatibleDC(NULL);

SelectObject(hCompatibleDC, hBitmap);

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

SelectObject(hCompatibleDC, GetStockObject(DC\_BRUSH));

SetDCBrushColor(hCompatibleDC, RGB(255, 0, 255));

if (!BitBlt(hCompatibleDC, 0, 0, bitmap.bmWidth, bitmap.bmHeight, hCompatibleDC, 0, 0, MERGECOPY))

{

return 1;

}

saveBitmap(hCompatibleDC, hBitmap, bitmap.bmWidth, bitmap.bmHeight);

InvalidateRect(hWnd, NULL, TRUE);

DeleteDC(hCompatibleDC);

return 0;

}

void runSetPixelFilter(HWND hwnd, HBITMAP hBitmap)

{

BITMAP bitmap;

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

HDC hCompatibleDC = CreateCompatibleDC(NULL);

SelectObject(hCompatibleDC, hBitmap);

for (int i = 0; i < bitmap.bmWidth; i++)

{

for (int j = 0; j < bitmap.bmHeight; j++)

{

SetPixel(hCompatibleDC, i, j, GetPixel(hCompatibleDC, i, j) & 0xFF00FF);

}

}

saveBitmap(hCompatibleDC, hBitmap, bitmap.bmWidth, bitmap.bmHeight);

InvalidateRect(hwnd, NULL, TRUE);

DeleteDC(hCompatibleDC);

}

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;

}

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;

}

void runTests(HWND hwnd)

{

HBITMAP cpy;

DWORD dwStartTime;

DWORD dwElapsed;

// open console

AllocConsole();

HANDLE stdHandle;

int hConsole;

FILE\* fp;

stdHandle = GetStdHandle(STD\_OUTPUT\_HANDLE);

hConsole = \_open\_osfhandle((long)stdHandle, \_O\_TEXT);

fp = \_fdopen(hConsole, "w");

freopen\_s(&fp, "CONOUT$", "w", stdout);

// measure BitBlt

printf("BitBlt: \n");

for (int i = 1; i <= 10; i++)

{

cpy = (HBITMAP)CopyImage(hBitmap, IMAGE\_BITMAP, i \* 200, i \* 200, 0);

dwStartTime = GetTickCount();

runBitBltFilter(hwnd, cpy);

dwElapsed = GetTickCount() - dwStartTime;

printf("[%4d x %4d] = %d.%d sec.\n", (i \* 200), (i \* 200), dwElapsed / 1000, dwElapsed - dwElapsed / 1000);

DeleteObject(cpy);

}

printf("\n");

// measure SetPixel/GetPixel

printf("Pixel: \n");

for (int i = 1; i <= 10; i++)

{

cpy = (HBITMAP)CopyImage(hBitmap, IMAGE\_BITMAP, i \* 200, i \* 200, 0);

dwStartTime = GetTickCount();

runSetPixelFilter(hwnd, cpy);

dwElapsed = GetTickCount() - dwStartTime;

printf("[%4d x %4d] = %d.%d sec.\n", (i \* 200), (i \* 200), dwElapsed / 1000, dwElapsed - dwElapsed / 1000);

DeleteObject(cpy);

}

printf("\nDone!");

}

**Часть 2. Оценка соотношения скорости работы функций.**

