Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные системы и системное программирование

Отчёт

к лабораторной работе

на тему

Расширенное использование оконного интерфейса Win 32 и GDI.

Формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, механизм перехвата сообщений (winhook)

Студент: гр.153502

Логвинович М.В.

Проверил: Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc147333809)

[2 Теоретические сведения 4](#_Toc147333810)

[3 Результат выполнения программы 5](#_Toc147333811)

[Список использованных источников 6](#_Toc147333812)

[Приложение А 7](#_Toc147333813)

# **1 ЦЕЛЬ РАБОТЫ**

Реализовать графическое приложение для анимации движения объектов с возможностью настройки траектории и скорости.

# **2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

GDI (Graphics Device Interface) — это часть Microsoft Windows API, предназначенной для работы с графикой и управления графическими ресурсами на уровне устройства. GDI предоставляет функции для рисования графических объектов, создания изображений, управления цветами и шрифтами, а также реализации различных эффектов. Этот компонент позволяет разработчикам создавать интерфейсы пользовательских приложений, включая окна, кнопки, текст и другие графические элементы. GDI обеспечивает визуальное качество и интерактивность приложений, делая их более привлекательными и функциональными для пользователей [1].

Чтобы приложения могли размещать выходные данные в памяти, а не отправлять их на фактическое устройство, используйте специальный контекст устройства для операций с растровыми изображениями, который называется контекстом устройства памяти. Контроллер памяти позволяет системе обрабатывать часть памяти как виртуальное устройство. Это массив битов в памяти, который приложение может временно использовать для хранения цветовых данных для растровых изображений, созданных на обычной поверхности рисования. Так как растровое изображение совместимо с устройством, контроллер памяти также иногда называют совместимым контекстом устройства [2].

Расширенное использование Win32 и GDI позволяет создавать более мощные и интерактивные Windows-приложения, предоставляя полный контроль над окнами, графикой и взаимодействием с пользователем. Эти теоретические сведения служат основой для разработки сложных и функциональных приложений под Windows.

# **3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ**

Результатом выполнения лабораторной работы является графическое приложение для анимации движения объектов с возможностью настройки траектории и скорости. Интерфейс графического приложения представляет собой главную область с рамкой, внутри которой происходит анимация объектов. Под областью анимации расположен ряд кнопок, позволяющих настроить скорость и траекторию движения объекта, приостановить или возобновить анимацию. Результат работы программы показан на рисунке 1.

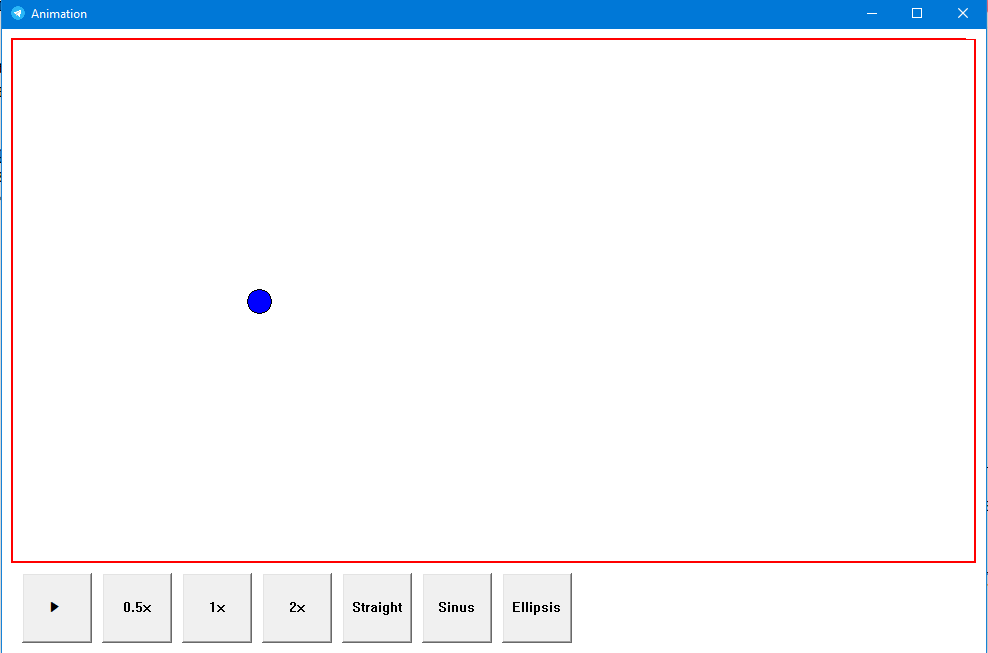


Рисунок 1 – Результат работы программы

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/gdi/windows-gdi

[2] [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/gdi/memory-device-contexts

# **ПРИЛОЖЕНИЕ А**

**Исходный код программы**

**Файл Source.cpp**

#include "windows.h"

#include "resource.h"

#include "cmath"

WNDCLASS MyRegisterClass(HINSTANCE hInstance);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

enum Path {

STRAIGHT, SINUS, ELLIPSIS

};

// global vars

RECT clientRect;

RECT object = {100, 250, 25, 25};

HBRUSH whiteBrush = CreateSolidBrush(RGB(255, 255, 255)); // White brush

HBRUSH grayBrush = CreateSolidBrush(RGB(127, 127, 127)); // Gray brush

HBRUSH blueBrush = CreateSolidBrush(RGB(0, 0, 255)); // Blue brush

bool forward = true;

bool isAnimating = false;

float speed = 1;

Path path = STRAIGHT;

HHOOK g\_hHook = NULL; // Hook for detecting 'Enter' clicks

#define BTN\_1 1

#define BTN\_2 2

#define BTN\_3 3

#define BTN\_4 4

#define BTN\_5 5

#define BTN\_6 6

#define BTN\_7 7

HWND btn1 = nullptr, btn2 = nullptr, btn3 = nullptr, btn4 = nullptr, btn5 = nullptr, btn6 = nullptr, btn7 = nullptr;

// Keyboard hook procedure(here just copy-paste logic for start animation button described below)

LRESULT CALLBACK KeyboardHookProc(int nCode, WPARAM wParam, LPARAM lParam) {

if (nCode >= 0) {

if (wParam == WM\_KEYDOWN) {

KBDLLHOOKSTRUCT\* kbData = (KBDLLHOOKSTRUCT\*)lParam;

if (kbData->vkCode == VK\_RETURN) {

// Enter key is pressed

isAnimating = !isAnimating;

if (isAnimating) {

SetWindowText(btn1, L"| |");

}

else {

SetWindowText(btn1, L"\u25B6");

}

}

}

}

return CallNextHookEx(g\_hHook, nCode, wParam, lParam);

}

void ClearPosition(HWND hWnd) {

isAnimating = false;

SetWindowText(GetDlgItem(hWnd, BTN\_1), L"\u25B6");

object = { 100, 250, 25, 25 };

speed = 1;

}

int counter = 0; // using for ellipsis animation

void moveArray(Path path = STRAIGHT) {

if (path == STRAIGHT) {

if (forward) object.left += 3;

else object.left -= 3;

if (object.left >= clientRect.right - clientRect.left - 50) forward = false;

if (object.left <= 0) forward = true;

object.right = object.left + 25;

object.bottom = object.top + 25;

}

else if (path == SINUS) {

double amplitude = 50.0; // Amplitude of the sinusoidal wave

double frequency = 0.05; // Frequency of the sinusoidal wave

// Calculate the new horizontal position based on the sine wave

if (forward) {

object.left += 3;

}

else {

object.left -= 3;

}

// Calculate the corresponding vertical position using the sine wave formula

object.top = static\_cast<LONG>(amplitude \* sin(frequency \* object.left)) + 195;

object.right = object.left + 25;

object.bottom = object.top + 25;

if (object.left >= clientRect.right - clientRect.left - 50) {

forward = false;

}

else if (object.left <= 0) {

forward = true;

}

}

else if (path == ELLIPSIS) {

double a = 400.0; // Horizontal radius

double b = 100.0; // Vertical radius

double angle = counter \* 0.05;

double x = a \* cos(angle);

double y = b \* sin(angle); // Parametric equation for an ellipse

object.left = static\_cast<LONG>(x) + (clientRect.right - clientRect.left) / 2;

object.top = static\_cast<LONG>(y) + (clientRect.bottom - clientRect.top) / 2;

// Update the ellipse's right and bottom coordinates

object.right = object.left + 25;

object.bottom = object.top + 25;

counter++;

if (counter >= 628) {

counter = 0;

}

}

}

void ShowRectangle(HDC dc) {

// selectobject - selects an object into the specified device context (DC), second parameter - hgdiobj (brush, pen, font)

// returns old object that new replaces

SelectObject(dc, grayBrush);

HDC memDC = CreateCompatibleDC(dc); // creating memory dc for specified dc

HBITMAP memBM = CreateCompatibleBitmap(dc, clientRect.right - clientRect.left - 20, clientRect.bottom - clientRect.top - 100); // creating bitmap in our dc with specified width and height

SelectObject(memDC, memBM);

HGDIOBJ oldBrush = SelectObject(memDC, whiteBrush);

HPEN pen = CreatePen(PS\_SOLID, 2, RGB(255,0,0));

HGDIOBJ oldPen = SelectObject(memDC, pen);

Rectangle(memDC, clientRect.left, clientRect.top, clientRect.right - 20, clientRect.bottom - 100);

SelectObject(memDC, oldBrush);

SelectObject(memDC, oldPen);

DeleteObject(pen);

oldBrush = SelectObject(memDC, blueBrush);

Ellipse(memDC, object.left, object.top, object.right, object.bottom);

SelectObject(memDC, oldBrush);

// transferring bits of color data from memDC to dc, (10,10) - upper left corner for dest rectangel; width and height of rectangles, source dc, upper-left corner in source dc, flag for direct copying

BitBlt(dc, 10, 10, clientRect.right - clientRect.left, clientRect.bottom - clientRect.top, memDC, 0,0, SRCCOPY);

DeleteDC(memDC);

DeleteObject(memBM);

}

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE, LPWSTR lpCmdLine, int nCmdShow)

{

// Setting hook (NULL for dll where we can describe hook procedure, 0 is thread id(default for connecting with all existing threads)

// WH\_KEYBOARD\_LL - low level keyboard detector(more recommended than WM\_KEYBOARD)

g\_hHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardHookProc, NULL, 0);

if (g\_hHook == NULL) {

MessageBox(NULL, L"Failed to install keyboard hook!", L"Error", MB\_ICONERROR);

return 1;

}

WNDCLASS wc = MyRegisterClass(hInstance);

if (!RegisterClass(&wc)) {

return EXIT\_FAILURE;

}

HWND hWnd = CreateWindow(wc.lpszClassName, L"Animation", WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 1000, 640, nullptr, nullptr, wc.hInstance, nullptr);

if (hWnd == INVALID\_HANDLE\_VALUE)

{

return EXIT\_FAILURE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

HDC dc = GetDC(hWnd); // get current window device context

MSG msg;

// Main message cycle:

while (1)

{

// check if message exist in queue, pm\_remove removes messages from queue after processing

if (PeekMessageA(&msg, NULL, 0, 0, PM\_REMOVE)) {

if (msg.message == WM\_QUIT) break;

TranslateMessage(&msg);

DispatchMessage(&msg);

}

else {

if (isAnimating) {

moveArray(path);

ShowRectangle(dc);

if (path == STRAIGHT) {

if (speed == 1) Sleep(10);

else if (speed == 0.5) Sleep(30);

}

else if (path == ELLIPSIS) {

if (speed == 1) Sleep(15);

else if (speed == 0.5) Sleep(30);

else Sleep(1);

}

else if (path == SINUS) {

if (speed == 1) Sleep(10);

else if (speed == 0.5) Sleep(25);

}

}

}

}

// Uninstall the keyboard hook when done

UnhookWindowsHookEx(g\_hHook);

return EXIT\_SUCCESS;

}

//

// ФУНКЦИЯ: MyRegisterClass()

//

// ЦЕЛЬ: Регистрирует класс окна.

//

WNDCLASS MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASS wcex{ sizeof(WNDCLASS) };

wcex.style = CS\_HREDRAW | CS\_VREDRAW; // vertical or horizontal redrawing

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_ICON1)); // makeintresource converts integer value to resource type

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW); // null for loading predefined cursor

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszClassName = L"CLASSNAME";

return wcex;

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case BTN\_1:

{

isAnimating = !isAnimating;

if (isAnimating) {

SetWindowText(btn1, L"| |");

}

else {

SetWindowText(btn1, L"\u25B6");

}

break;

}

case BTN\_2:

{

speed = 0.5;

break;

}

case BTN\_3:

{

speed = 1;

break;

}

case BTN\_4:

{

speed = 2;

break;

}

case BTN\_5:

{

path = STRAIGHT;

ClearPosition(hWnd);

break;

}

case BTN\_6:

{

path = SINUS;

ClearPosition(hWnd);

break;

}

case BTN\_7:

{

path = ELLIPSIS;

ClearPosition(hWnd);

break;

}

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

case WM\_SIZE:

GetClientRect(hWnd, &clientRect); // get window's client area coordinates; clientRect - data receiver

// WS\_VISIBLE - show button

if (btn1 != nullptr) {

DestroyWindow(btn1);

btn1 = nullptr;

}

if (btn2 != nullptr) {

DestroyWindow(btn2);

btn2 = nullptr;

}

if (btn3 != nullptr) {

DestroyWindow(btn3);

btn3 = nullptr;

}

if (btn4 != nullptr) {

DestroyWindow(btn4);

btn4 = nullptr;

}

if (btn5 != nullptr) {

DestroyWindow(btn5);

btn5 = nullptr;

}

if (btn6 != nullptr) {

DestroyWindow(btn6);

btn6 = nullptr;

}

if (btn7 != nullptr) {

DestroyWindow(btn7);

btn7 = nullptr;

}

btn1 = CreateWindow(L"BUTTON", L"\u25B6", WS\_VISIBLE | WS\_CHILD, clientRect.left + 20, clientRect.bottom - 80, 70, 70,

hWnd, reinterpret\_cast<HMENU>(BTN\_1), NULL, NULL);

btn2 = CreateWindow(L"BUTTON", L"0.5x", WS\_VISIBLE | WS\_CHILD, clientRect.left + 100, clientRect.bottom - 80, 70, 70,

hWnd, reinterpret\_cast<HMENU>(BTN\_2), NULL, NULL);

btn3 = CreateWindow(L"BUTTON", L"1x", WS\_VISIBLE | WS\_CHILD, clientRect.left + 180, clientRect.bottom - 80, 70, 70,

hWnd, reinterpret\_cast<HMENU>(BTN\_3), NULL, NULL);

btn4 = CreateWindow(L"BUTTON", L"2x", WS\_VISIBLE | WS\_CHILD, clientRect.left + 260, clientRect.bottom - 80, 70, 70,

hWnd, reinterpret\_cast<HMENU>(BTN\_4), NULL, NULL);

btn5 = CreateWindow(L"BUTTON", L"Straight", WS\_VISIBLE | WS\_CHILD, clientRect.left + 340, clientRect.bottom - 80, 70, 70,

hWnd, reinterpret\_cast<HMENU>(BTN\_5), NULL, NULL);

btn6 = CreateWindow(L"BUTTON", L"Sinus", WS\_VISIBLE | WS\_CHILD, clientRect.left + 420, clientRect.bottom - 80, 70, 70,

hWnd, reinterpret\_cast<HMENU>(BTN\_6), NULL, NULL);

btn7 = CreateWindow(L"BUTTON", L"Ellipsis", WS\_VISIBLE | WS\_CHILD, clientRect.left + 500, clientRect.bottom - 80, 70, 70,

hWnd, reinterpret\_cast<HMENU>(BTN\_7), NULL, NULL);

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps); // handle to context device

HPEN pen = CreatePen(PS\_SOLID, 2, RGB(255, 0, 0));

HGDIOBJ oldPen = SelectObject(hdc, pen);

Rectangle(hdc, clientRect.left + 10, clientRect.top + 10, clientRect.right - 20, clientRect.bottom - 90);

EndPaint(hWnd, &ps);

break;

}

break;

default:

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

}

return 0;

}