УО «Белорусский государственный технологический университет»

Факультет информационных технологий

Кафедра «Информационных систем и технологий»

Специальность 1-40 05 01 «Информационные Системы и Технологии»

**Реферат**

**На тему "Рисование графиков на С++ при помощи SDL"**

|  |  |  |  |
| --- | --- | --- | --- |
| **Выполнил** |  |  |  |
| Студент 1 курса группы 1 |  |  | Д.И. Велютич |
|  | подпись, дата |  | инициалы и фамилия |
| **Проверил** |  |  | Н.И. Белодед |
|  | подпись, дата |  | инициалы и фамилия |

Минск 2023

Содержание реферата

[1. Введение 2](#_Toc132593463)

[2. Теория 2](#_Toc132593464)

[3. Реализация библиотеки для работы с графиками 3](#_Toc132593465)

[4. Создание программы для визуализации графиков на экране 7](#_Toc132593466)

[Скриншот и видео работы программы: 14](#_Toc132593467)

[Заключение: 14](#_Toc132593468)

# 1. Введение

В данном реферате рассмотрен процесс создания приложения на языке программирования C++ для работы с графиками на оси координат (X and Y)

Для своей работы программа использует ряд библиотек, о которых также будет написано в реферате. Работает программа благодаря популярной библиотеке SDL, с помощью которой выводится изображение.

# 2. Теория

SDL (Simple DirectMedia Layer) - это кроссплатформенная библиотека, которая предоставляет доступ к низкоуровневым функциям работы с аппаратным обеспечением компьютера, такими как аудио, видео, клавиатура, мышь, джойстики и другие устройства ввода-вывода. Она была создана для упрощения разработки мультимедийных приложений и компьютерных игр, позволяя программистам обращаться к аппаратному обеспечению через простой и понятный интерфейс.

SDL поддерживает несколько операционных систем, включая Windows, MacOS, Linux, iOS, Android, а также несколько аппаратных платформ, таких как PC, игровые консоли, мобильные устройства и другие.

SDL позволяет работать с графикой через библиотеки OpenGL и Direct3D, что позволяет создавать трехмерные графические приложения и игры, используя аппаратное ускорение. Библиотека также имеет возможность работать с 2D-графикой, предоставляя функции для рисования линий, примитивов и текстур.

Одной из наиболее полезных функций SDL является управление аудиоустройствами. SDL позволяет программисту создавать звуковые эффекты и музыку, а также управлять уровнем громкости и каналами звука.

Кроме того, SDL имеет функции для работы с клавиатурой, мышью и джойстиками, которые позволяют программистам создавать интерактивные приложения и игры. С помощью библиотеки можно обрабатывать нажатия клавиш, движения мыши и джойстика, что позволяет создавать различные интерфейсы для пользователей.

Таким образом, SDL является мощной библиотекой для разработки мультимедийных приложений и игр, которая позволяет программистам работать с различными устройствами и операционными системами, используя простой и удобный интерфейс.

# 3. Реализация библиотеки для работы с графиками

Для начала создадим библиотеку для корректного отображения графического интерфейса программы путём создания фалов ***graph.cpp***, ***graph.h***, ***raster\_graphics.h***, ***console\_render.cpp***, ***console\_render.h***, а также ещё несколько неважных файлов для отладки мою текущей программы.  
  
***graph.h***

// Заголовочный файл для класса графа, содержащий объявление методов и полей класса.

#pragma once

#include <vector>

#include <string>

class Graph {

public:

// Добавить точку в граф с заданными координатами (x, y)

void addPoint(int x, int y);

// Соединить все точки в графе линиями, образуя график

void connectPoints();

// Сохранить граф в файл в формате "x y\nx y\n...\n---\n"

// где каждая точка задана координатами (x, y) в одной строке,

// а символ "---" используется в качестве разделителя между графами в файле

void saveToFile(const std::string& filepath) const;

// Загрузить граф из файла в формате, описанном выше

void loadFromFile(const std::string& filepath);

// Получить вектор точек, представляющих граф

const std::vector<std::pair<int, int>>& getPoints() const;

private:

std::vector<std::pair<int, int>> points; // Вектор точек, представляющих граф

};

***graph.cpp***

#include "graph.h"

#include <fstream>

void Graph::addPoint(int x, int y) {

points.emplace\_back(x, y); // Добавление точки в вектор

}

void Graph::connectPoints() {

// Реализовать логику соединения точек прямыми линиями.

// Для программы на консоли это может быть достигнуто путем вычисления

// линейных сегментов и обновления буфера консоли символами линии.

}

void Graph::saveToFile(const std::string& filepath) const {

std::ofstream outfile(filepath, std::ios::app); // Открыть файл в режиме добавления информации в конец файла

if (!outfile.is\_open()) { // Проверка на открытие файла

throw std::runtime\_error("Не удалось открыть файл для записи"); // Вывод исключения, если файл не был открыт

}

for (const auto& point : points) { // Цикл по точкам графика

outfile << point.first << ' ' << point.second << std::endl; // Запись координат точек в файл

}

outfile << "---" << std::endl; // Разделитель между графиками

outfile.close(); // Закрыть файл

}

void Graph::loadFromFile(const std::string& filepath) {

std::ifstream infile(filepath); // Открыть файл на чтение

if (!infile.is\_open()) { // Проверка на открытие файла

throw std::runtime\_error("Не удалось открыть файл для чтения"); // Вывод исключения, если файл не был открыт

}

points.clear(); // Очистить вектор точек

int x, y;

while (infile >> x >> y) { // Цикл по строкам в файле

points.emplace\_back(x, y); // Добавить точку в вектор точек

}

infile.close(); // Закрыть файл

}

const std::vector<std::pair<int, int>>& Graph::getPoints() const {

return points; // Вернуть вектор точек графика

}

***Raster\_graphics.h***

#pragma once

#include "graph.h"

// Отрисовка координатных осей

void drawAxis();

// Отрисовка графика

void drawGraph(const Graph& graph);

// Отрисовка графика с подписями точек

void drawGraphWithLabels(const Graph& graph);

// Создание графика

Graph createGraph();

// Добавление точки в график

void addPointToGraph(Graph& graph, int x, int y);

// Соединение точек в графике

void connectPointsInGraph(Graph& graph);

// Сохранение графика в файл

void saveGraphToFile(const Graph& graph, const std::string& filepath);

// Загрузка графика из файла

void loadGraphFromFile(Graph& graph, const std::string& filepath);

***console\_renderer.h***

#pragma once

#include "graph.h"

class ConsoleRenderer {

public:

void drawAxis(const Graph& graph); // Добавляем здесь параметр graph

void drawGraph(const Graph& graph);

};

***console\_renderer.cpp***

#include "console\_renderer.h"

#include <iostream>

#include <iomanip>

#include <vector>

#include <algorithm>

const int WIDTH = 50;

const int HEIGHT = 20;

void ConsoleRenderer::drawAxis(const Graph& graph) {

// Нарисовать координатные оси используя простой текстовый вывод.

for (int y = 0; y < HEIGHT; ++y) {

for (int x = 0; x < WIDTH; ++x) {

if (x == WIDTH / 2) {

std::cout << '|';

}

else if (y == HEIGHT / 2) {

std::cout << '-';

}

else {

std::cout << ' ';

}

}

std::cout << std::endl;

}

}

void ConsoleRenderer::drawGraph(const Graph& graph) {

// Создать 2D-массив, представляющий пространство консоли

std::vector<std::vector<char>> consoleSpace(HEIGHT, std::vector<char>(WIDTH, ' '));

// Нарисовать координатные оси в 2D-массиве

for (int y = 0; y < HEIGHT; ++y) {

for (int x = 0; x < WIDTH; ++x) {

if (x == WIDTH / 2) {

consoleSpace[y][x] = '|';

}

else if (y == HEIGHT / 2) {

consoleSpace[y][x] = '-';

}

}

}

// Нарисовать точки графика в 2D-массиве

const auto& points = graph.getPoints();

for (const auto& point : points) {

int x = WIDTH / 2 + point.first;

int y = HEIGHT / 2 - point.second;

if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT) {

consoleSpace[y][x] = 'o';

}

}

// Нарисовать линии между точками в 2D-массиве

// Можно использовать алгоритм рисования линий, например, Алгоритм Брезенхэма

// Вывести содержимое 2D-массива в консоль

for (const auto& row : consoleSpace) {

for (char c : row) {

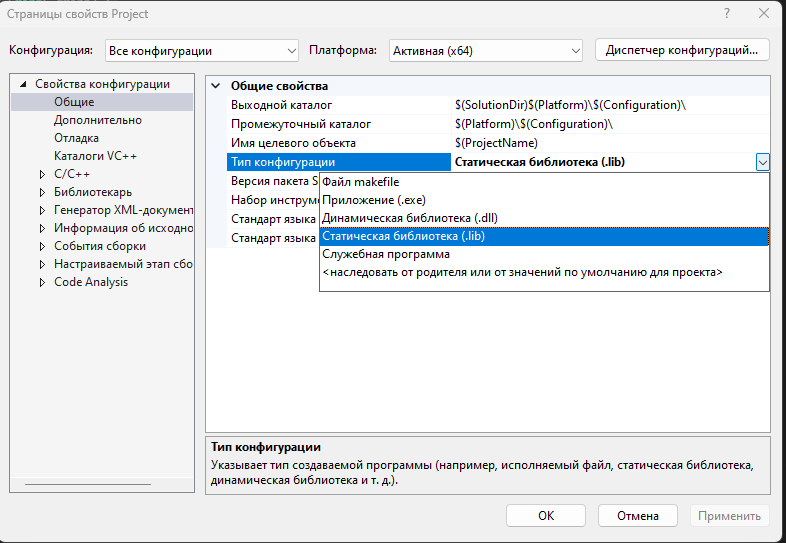
std::cout << c;

}

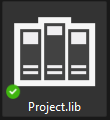
std::cout << std::endl;

}

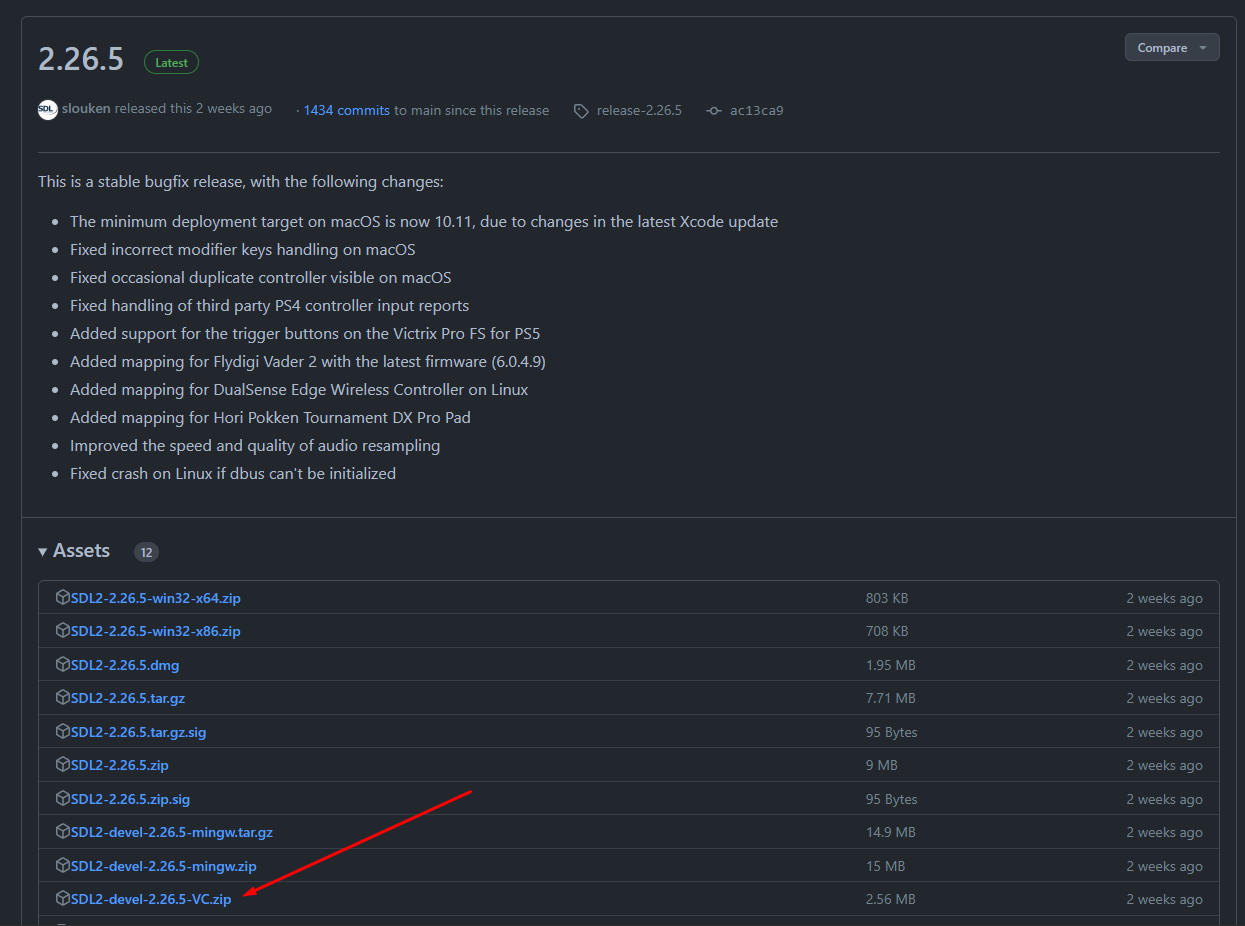
}

Затем меняем в настройках проекта тип конфигурации с приложения *.exe* на статическая библиотека *.lib*  


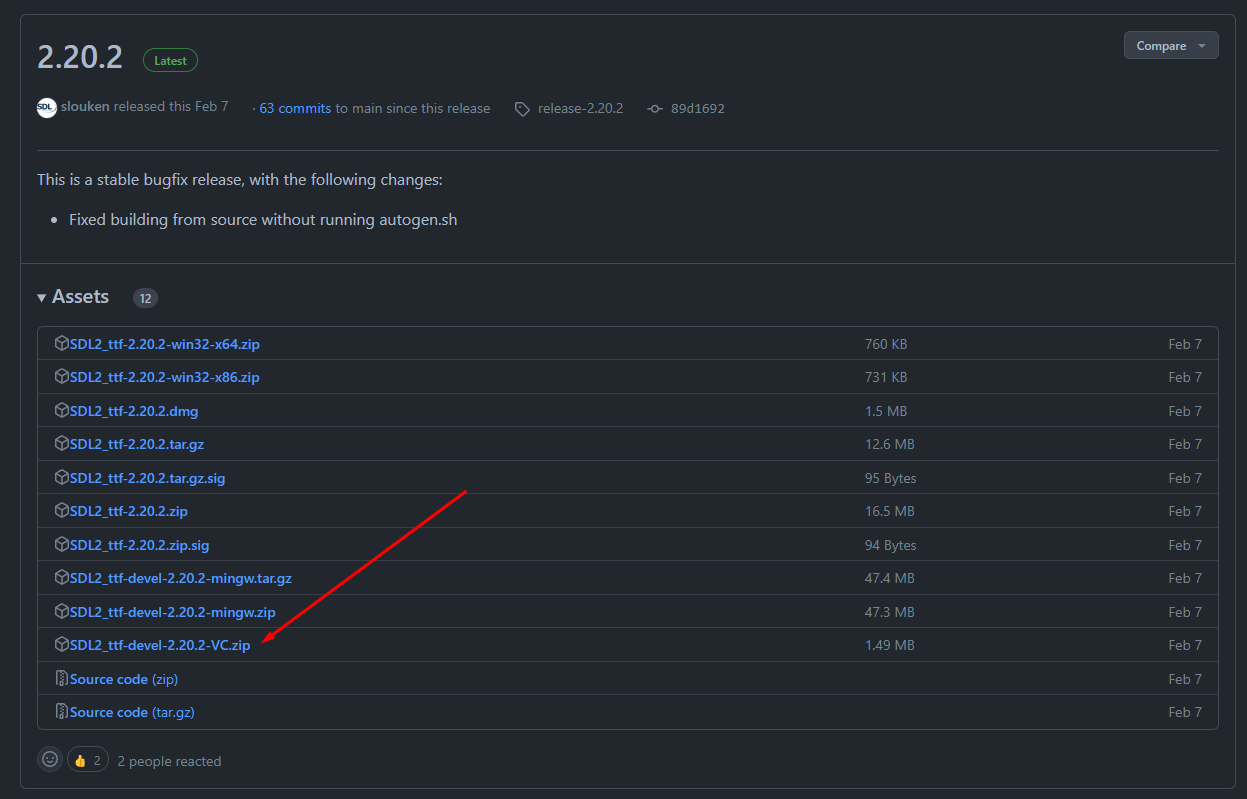
Потом собираем проект и получаем библиотеку ***Project.lib*** (в моём случае).



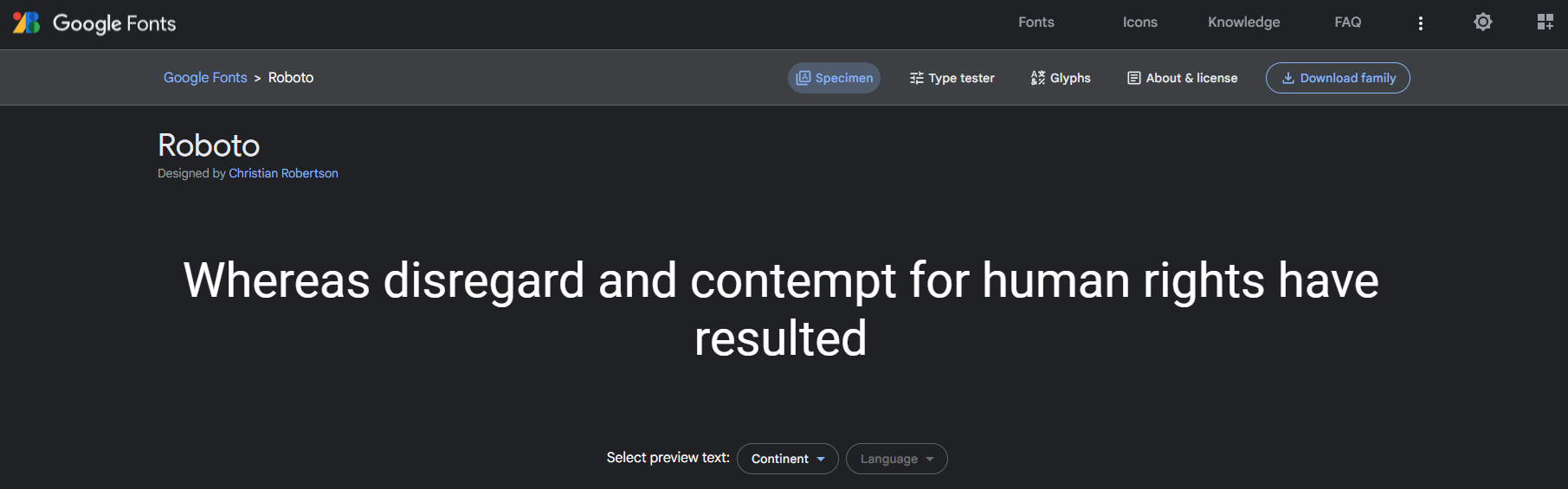
# 4. Создание программы для визуализации графиков на экране

Создаём новый проект в Visual Studio, над которым надо будет провернуть некоторые манипуляции:  
1) Скачиваем библиотеки SDL2 c GitHub (Developer version for VS)  


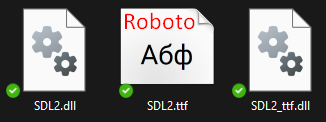
2) Скачиваем библиотеки SDL2\_TTF c GitHub (Developer version for VS)



3) Скачиваем любой понравившийся шрифт в формате *.ttf* для корректного отображения шрифтов (я выбрал семейство шрифтов Roboto с Google Fonts)



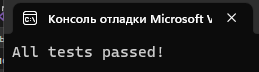
4) Так как я работаю из-под Windows, нужно загрузить *.dll* файлы SDL2 и SDL2\_TTF с вышеупомянутых репозиториев, для корректной работы программы.



5) Подключаем все вышеупомянутые компоненты (в том числе и библиотеку из прошлого проекта) к новому проекту.   
  
*.DLL* файлы закидываем к исходным файлам в проект;  
Библиотеки закидываем через компоновщика в программу



**Готово**! Нужные зависимости успешно установлены.  
  
Если первой подключить библиотеку Project.lib (из прошлого проекта), а далее запустить, увидим сообщение о том, что все тесты пройдены и программа работает исправно.



Далее создаем 3 файла: ***main.cpp***, ***sdl\_renderer.cpp***, ***sdl\_renderer.h***.  
  
***sdl\_renderer.h***  
  
// заголовочный файл SDLRenderer, объявляющий класс SDLRenderer и его методы

#pragma once

#include "graph.h"

#include <SDL.h>

#include <SDL\_ttf.h>

class SDLRenderer {

public:

// конструктор, который принимает SDL\_Renderer

explicit SDLRenderer(SDL\_Renderer\* renderer);

// методы отрисовки осей и графика

void drawAxis();

void drawGraph(const Graph& graph);

// метод отрисовки подписанных осей

void drawLabeledAxis();

private:

// переменная типа SDL\_Renderer, используемая для отрисовки графики

SDL\_Renderer\* renderer;

// указатель на шрифт, используемый для подписи осей

TTF\_Font\* font;

};

***sdl\_renderer.cpp***

#include "sdl\_renderer.h"

#include <iostream>

SDLRenderer::SDLRenderer(SDL\_Renderer\* renderer)

: renderer(renderer) {

font = TTF\_OpenFont("SDL2.ttf", 16);

if (!font) {

std::cerr << "Не удалось загрузить шрифт: " << TTF\_GetError() << std::endl;

}

}

void SDLRenderer::drawAxis() {

// Получаем размер окна

int windowWidth, windowHeight;

SDL\_GetRendererOutputSize(renderer, &windowWidth, &windowHeight);

// Устанавливаем цвет черный

SDL\_SetRenderDrawColor(renderer, 0, 0, 0, 255);

// Рисуем горизонтальную ось

SDL\_RenderDrawLine(renderer, 0, windowHeight / 2, windowWidth, windowHeight / 2);

// Рисуем вертикальную ось

SDL\_RenderDrawLine(renderer, windowWidth / 2, 0, windowWidth / 2, windowHeight);

// Можно добавить код здесь, чтобы нарисовать подписи осей с помощью расширения SDL\_ttf, если это требуется

}

void SDLRenderer::drawGraph(const Graph& graph) {

// Устанавливаем цвет для графика другим цветом (например, красным)

SDL\_SetRenderDrawColor(renderer, 0, 0, 255, 255);

// Получаем размер окна

int windowWidth, windowHeight;

SDL\_GetRendererOutputSize(renderer, &windowWidth, &windowHeight);

// Рисуем точки графика

const auto& points = graph.getPoints();

for (const auto& point : points) {

int x = windowWidth / 2 + point.first;

int y = windowHeight / 2 - point.second;

SDL\_RenderDrawPoint(renderer, x, y);

}

// Рисуем линии между последовательными точками

for (size\_t i = 1; i < points.size(); ++i) {

int x1 = windowWidth / 2 + points[i - 1].first;

int y1 = windowHeight / 2 - points[i - 1].second;

int x2 = windowWidth / 2 + points[i].first;

int y2 = windowHeight / 2 - points[i].second;

SDL\_RenderDrawLine(renderer, x1, y1, x2, y2);

}

}

***main.cpp***

#include <iostream>

#include <SDL.h>

#include <SDL\_ttf.h>

#include "sdl\_renderer.h"

#include "graph.h"

void displayMenu() {

system("cls");

std::cout << "Выберите действие:\n";

std::cout << "1) Импортировать точки из файла\n";

std::cout << "2) Добавить точки на график\n";

std::cout << "3) Отображение финального графика\n";

std::cout << "4) Экспортировать данные в файл\n";

std::cout << "5) Вывод текущих точек на экран\n";

std::cout << "6) Выход из программы\n";

std::cout << "\nВаш выбор: ";

}

int main(int argc, char\* argv[]) {

// Инициализируем SDL

setlocale(0, "rus");

if (SDL\_Init(SDL\_INIT\_VIDEO) < 0) { // Инициализация SDL

std::cerr << "Не удалось инициализировать SDL: " << SDL\_GetError() << std::endl;

return 1;

}

// Инициализируем SDL\_ttf

if (TTF\_Init() < 0) { // Инициализация SDL\_ttf

std::cerr << "Не удалось инициализировать SDL\_ttf: " << TTF\_GetError() << std::endl;

return 1;

}

int screenWidth, screenHeight;

std::cout << "Введите ширину экрана: ";

std::cin >> screenWidth;

std::cout << "Введите высоту экрана: ";

std::cin >> screenHeight;

// Создаем окно

SDL\_Window\* window = SDL\_CreateWindow("Grafics by Velutich", SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED, screenWidth, screenHeight, SDL\_WINDOW\_SHOWN); // Название окна и его параметры

if (!window) {

std::cerr << "Не удалось создать окно: " << SDL\_GetError() << std::endl;

return 1;

}

// Создаем рендерер

SDL\_Renderer\* renderer = SDL\_CreateRenderer(window, -1, SDL\_RENDERER\_ACCELERATED | SDL\_RENDERER\_PRESENTVSYNC);

if (!renderer) {

std::cerr << "Не удалось создать рендерер: " << SDL\_GetError() << std::endl;

return 1;

}

// Создаем экземпляр SDLRenderer

SDLRenderer sdlRenderer(renderer);

// Создаем экземпляр Graph и добавляем несколько точек

Graph graph;

bool running = true;

while (running) {

displayMenu();

int choice;

std::cin >> choice;

switch (choice) {

case 1: {

// Запросить у пользователя название файла

std::cout << "Введите название файла: ";

std::string fileName;

std::cin >> fileName;

// Загрузить данные о точках графика из файла

try {

graph.loadFromFile(fileName);

std::cout << "Данные успешно загружены из файла " << fileName << std::endl;

}

catch (const std::runtime\_error& e) {

std::cerr << "Ошибка: " << e.what() << std::endl;

}

system("pause");

break;

}

case 2: {

int numPoints;

std::cout << "Сколько точек вы хотите добавить на график? ";

std::cin >> numPoints;

double x, y;

for (int i = 0; i < numPoints; i++) {

std::cout << "Введите координату x для точки " << i + 1 << ": ";

std::cin >> x;

std::cout << "Введите координату y для точки " << i + 1 << ": ";

std::cin >> y;

// Проверяем, чтобы координаты были в пределах размеров экрана

if (x < -screenWidth || x > screenWidth || y < -screenHeight || y > screenHeight) {

std::cerr << "Координаты вне пределов экрана. Попробуйте снова.\n";

i--; // Повторяем ввод для этой точки

}

else {

graph.addPoint(x, y);

}

}

break;

}

case 3: {

std::cout << "Вы можете нажать ESC находясь в окне графика для возварата в меню" << std::endl;

bool displayGraph = true;

while (displayGraph) {

SDL\_Event e;

while (SDL\_PollEvent(&e)) {

if (e.type == SDL\_QUIT) {

displayGraph = false;

running = false; // Выход из программы

}

else if (e.type == SDL\_KEYDOWN) {

if (e.key.keysym.sym == SDLK\_ESCAPE) { // Если нажата клавиша ESC

displayGraph = false; // Возвращаемся в главное меню

}

else if (e.key.keysym.sym == SDLK\_q) { // Если нажата клавиша Q

displayGraph = false;

running = false; // Выход из программы

}

}

}

// Очищаем экран

SDL\_SetRenderDrawColor(renderer, 255, 255, 255, 255);

SDL\_RenderClear(renderer);

// Рисуем координатные оси и график

sdlRenderer.drawAxis();

sdlRenderer.drawGraph(graph);

// Обновляем экран

SDL\_RenderPresent(renderer);

}

break;

}

case 4: {

// Запросить у пользователя название файла

std::cout << "Введите название файла: ";

std::string fileName;

std::cin >> fileName;

// Сохранить данные о точках графика в файл

try {

graph.saveToFile(fileName);

std::cout << "Данные успешно сохранены в файл " << fileName << std::endl;

}

catch (const std::runtime\_error& e) {

std::cerr << "Ошибка: " << e.what() << std::endl;

}

system("pause");

break;

}

case 6: {

running = false;

break;

}

case 5: {

// Вывести точки на экран

const auto& points = graph.getPoints();

if (points.empty()) {

std::cout << "График пуст, нет точек для вывода." << std::endl;

}

else {

std::cout << "Текущие точки графика:" << std::endl;

for (const auto& point : points) {

std::cout << point.first << ", " << point.second << std::endl;

}

}

system("pause");

break;

}

default: {

std::cout << "Неизвестный выбор. Попробуйте снова.\n";

}

}

}

// Чистим память

TTF\_Quit();

SDL\_DestroyRenderer(renderer);

SDL\_DestroyWindow(window);

SDL\_Quit();

return 0;

}

Далее запускаем программу, вводим нужную нам ширину и высоту экрана (на примере 1280x720px). Видим 6 пунктов:  
  
1) Импортировать точки из файла

2) Добавить точки на график

3) Отображение финального графика

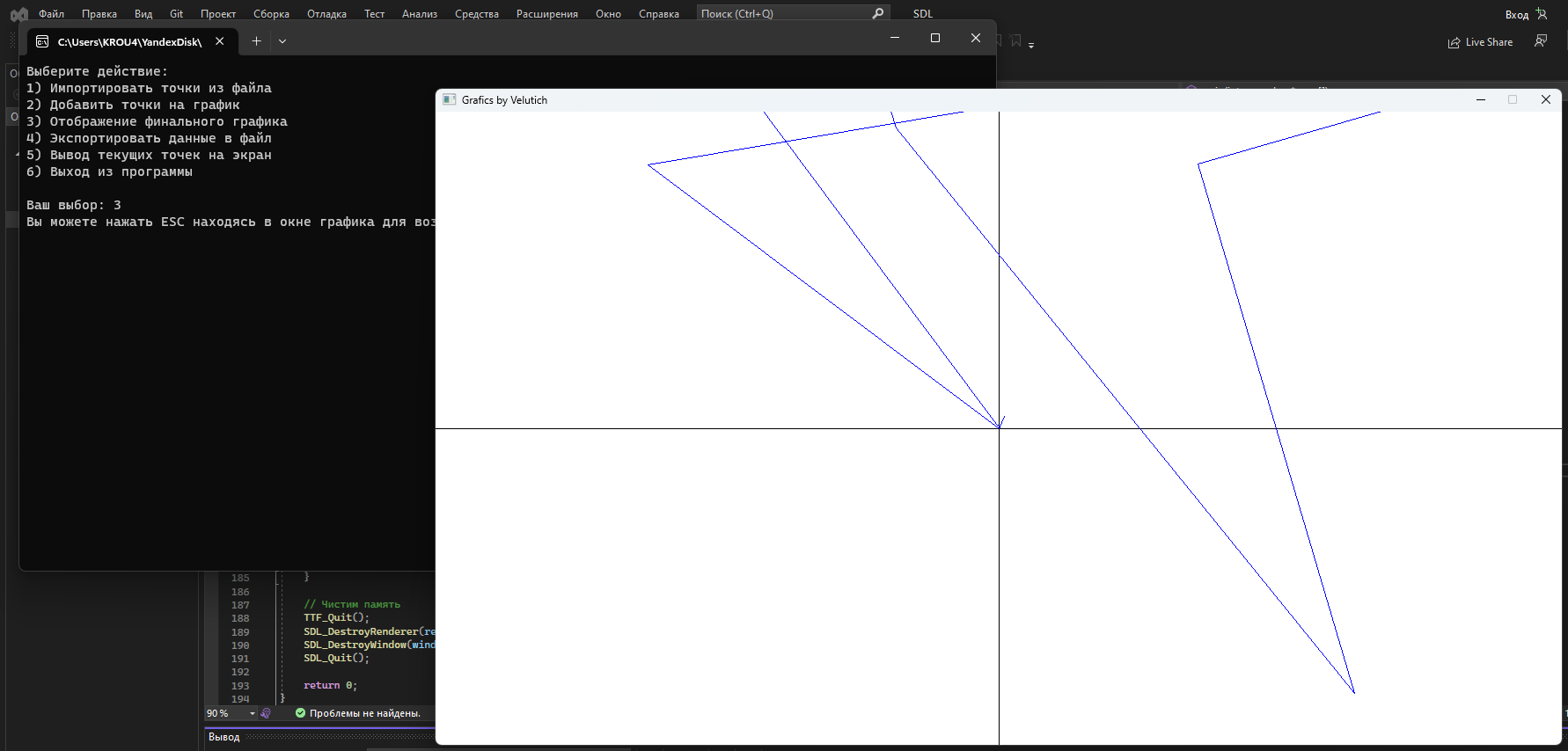
4) Экспортировать данные в файл

5) Вывод текущих точек на экран

6) Выход из программы

Код этих функций прописан в файле main.cpp под соответствующими номера свич-кейса.

Суть работы каждого из пунктов лежит в названии пункта)

Скриншот и видео работы программы:  
  
  


\*Если видео из Word’a открыть не получается, то вот ссылка на **это же** видео на площадке YouTube: <https://youtu.be/dTCMhG7MVw8>

Заключение:  
В этом реферате мы, по сути, с нуля разработали программу для работы с графиками на языке С++.  
SDL2 является отличной библиотекой для визуализации, с помощью которой можно разрабатывать игры и даже игровые движки, а работа с графиками является лишь верхушкой айсберга.