МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: «Обработка изображений»

Студент гр. 0304 _____ Максименко Е.М.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Максименко Е.М.

Группа 0304

Тема работы: «Обработка изображений»

Исходные данные:
Программа должна считывать и записывать изображения в формате PNG. Изображение должно обрабатываться с помощью заданных условием функций. Программа должна иметь графический интерфейс. При записи изображения в файл строки, по необходимости, должны использовать выравнивание.
Содержание пояснительной записки:
«Содержание», «Введение», «Считывание и запись изображения», «Обработка изображения», «Графический интерфейс»,«Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 16 страниц.

Дата выдачи задания: 05.04.2021

Дата сдачи реферата:

Дата защиты реферата:

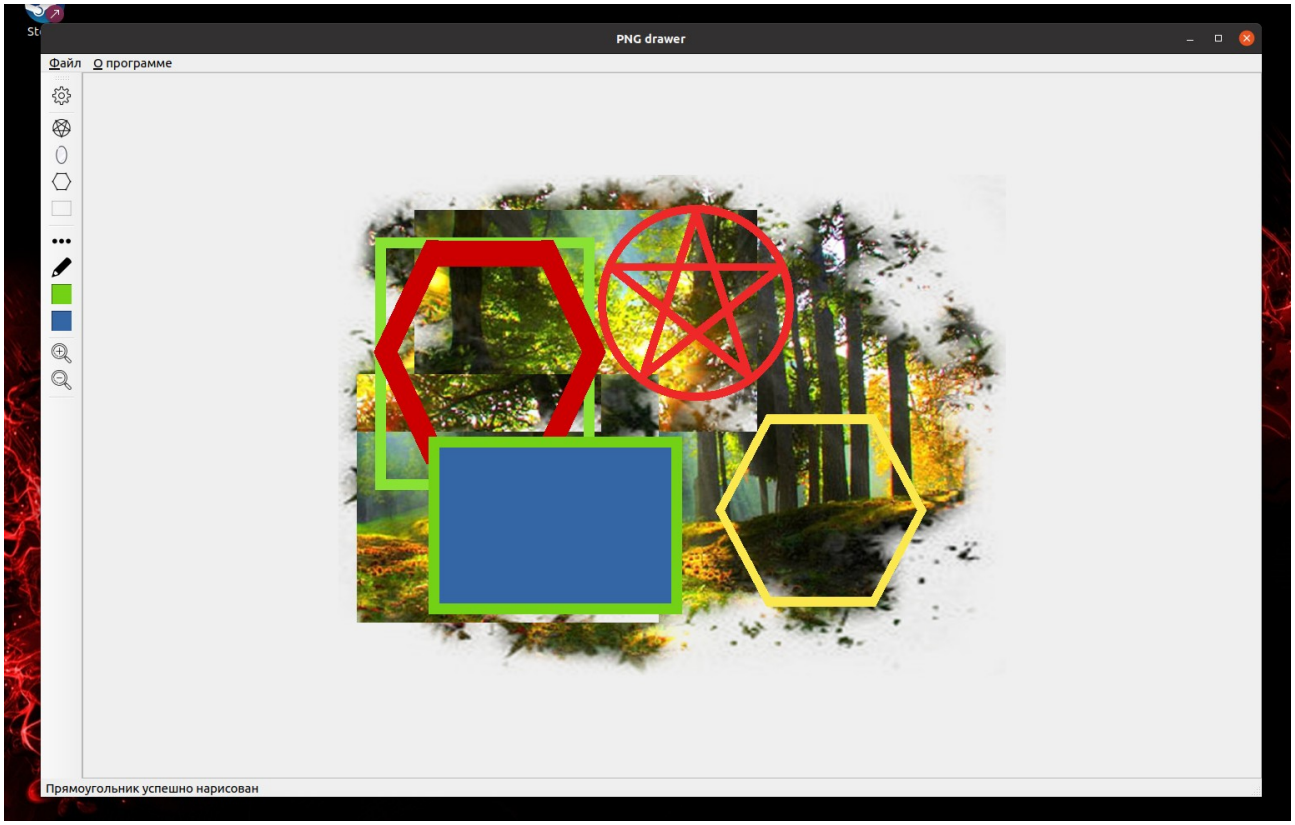
Студент _____ Максименко Е.М,

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

Была поставлена задача создать приложение, которое будет обладать функционалом считывания изображения в формате PNG, его записью в файл, а также функциями для обработки данного изображения.

Программа состоит из 3-х файлов исходного кода и 2-х заголовочных файлов. Таким образом была разделена логика взаимодействия с пользователем (графический интерфейс) и логика работы с изображениями (считывание, запись, обработка изображений в формате PNG).



Демонстрация работы программы

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1 Цель курсовой работы и исходные условия

Целью данной работы является создание программы, соответствующей условию задания:

Условие задания:

- **Формат картинки PNG (рекомендуем использовать библиотеку libpng)**
- файл всегда соответствует формату PNG
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- все поля стандартных PNG заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна реализовывать следующий функционал по обработке PNG-файла:

1. Отражение заданной области. Этот функционал определяется:
    - выбором оси относительно которой отражать (горизонтальная или вертикальная)
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области
2. Рисование пентаграммы в круге. Пентаграмма определяется:
    - **либо** координатами левого верхнего и правого нижнего угла квадрата, в который вписана окружность пентаграммы, **либо** координатами ее центра и радиусом окружности
    - толщиной линий и окружности
    - цветом линий и окружности
3. Рисование прямоугольника. Он определяется:
    - Координатами левого верхнего угла
    - Координатами правого нижнего угла
    - Толщиной линий
    - Цветом линий
    - Прямоугольник может быть залит или нет
    - цветом которым он залит, если пользователем выбран залитый
4. Рисование правильного шестиугольника. Шестиугольник определяется:
    - **либо** координатами левого верхнего и правого нижнего угла квадрата, в который он вписан, **либо** координатами его центра и радиусом в который он списан
    - толщиной линий
    - цветом линий
    - шестиугольник может быть залит или нет
    - цветом которым залит шестиугольник, если пользователем выбран залитый

1.2 Задачи

Для достижения поставленной цели требуется решить следующие задачи:

- Создание файловой структуры программы — разбиение программы на подпрограммы для разделения логики работы с изображением и логики работы с пользователем.
- Разработка программного кода, выполняющего необходимые задачи.
- Написание *Makefile* для сборки программы.
- Сборка и тестирование программы.

2. СЧИТЫВАНИЕ И ЗАПИСЬ ИЗОБРАЖЕНИЯ

2.1 Класс для хранения изображения

Для считывания, хранения, записи изображения и его обработки в заголовчном файле *pngimage.h* объявлен класс *PNGImage*, а также для работы с пикселями объявлена структура *PixelPainted*. В каждом методе под именем файла подразумевается полный путь до него.

Класс *PNGImage* имеет публичные поля *png_int_32 width*, *png_int_32 height*, которые отвечают за параметры ширины и высоты изображения, *png_byte color_type*, *png_byte bit_depth*, которые отвечают за параметры цветовой схемы и глубины цвета изображения, а также *struct stat file_info*, которое отвечает за хранение информации об изображении, такой как время последнего изменения, размер файла и т.д. Также присутствуют объявления публичных методов класса, таких как *read_image()*, *write_image()*, отвечающих за считывание и запись изображения, метод *get_file_info()*, отвечающий за получение информации о файле, а также методы обработки изображения: *mirror()*, *draw_pentagram()*, *draw_rectangle()*, *draw_hexagon()*. В классе объявлены приватные поля *png_structp png_ptr*, *png_infop info_ptr*, *png_infop end_ptr*, *png_bytep *rows*, отвечающие за хранения структуры изображения, служащей для считывания и записи изображения, хранение заголовков и карты пикселей. Также объявлено поле *int bytes_per_pixel*, отвечающее за хранения размера одного пикселя в байтах для более удобной работы с картой пикселей.

В классе объявлены приватные методы *create_structs(), init_io(), __read_image(), __write_image(), __add_trash_data()*, являющиеся вспомогательными для считывания и записи файла, а также вспомогательные методы обработки изображения: *_draw_line(), _normalize_coords(), _set_pixel(), _draw_circle(), _draw_circle_thickness()*.

2.2 Считывание из файла

Для чтения изображения из файла были разработаны метод класса *PNGImage void read_image(const char *filename)* и вспомогательные методы *void create_structs(FILE *file), void init_io(FILE *file), void __read_image(FILE *file)*.

Основной метод *read_image()* получает как аргумент имя файла и пытается его открыть на чтение в бинарном режиме. В случае ошибки выбрасывается исключение. После открытия файла последовательно вызываются методы *create_structs(), init_io(), __read_image(),* в качестве параметра они получают дескриптор ранее открытого файла. После выполнения кода данных методов вызывается функция *png_destroy_read_struct()*, которая очистит *png_ptr*, потом вызывается метод получения информации о файле *get_file_info()*, после чего файл закрывается.

Метод *create_struct()* оперирует с полученным файловым дескриптором. Сперва метод проверяет сигнатуру файла на то, совпадает ли она с сигнатурой PNG изображения. В случае несовпадения будет выброшено исключение. После этого создаются структуры *png_ptr*, *info_ptr*, *end_ptr*, и, в случае неудачи, выбрасывается исключение и закрывается файл.

Метод *inti_io()* получает на вход файловый дескриптор. Сперва инициализируется структура *png_ptr*, после чего считывается заголовок *IHDR* изображения. После этого производятся необходимые манипуляции с форматом изображения, такие как принудительный перевод к глубине цвета, равной 8, а также задается значение поля *bytes_per_pixel* в зависимости от цветовой схемы изображения. После данных манипуляций структура *info_ptr*, хранящая поля заголовка *IHDR*, обновляется.

Метод *__read_image()* получает в качестве аргумента файловый дескриптор и уже считывает карту пикселей изображения в массив *rows*. Также данным методом считывается заголовок *IEND* изображения.

2.3 Запись в файл

Для записи изображения в файл были разработаны метод класса *PNGImage void write_image(const char *filename)* и вспомогательные методы *void __write_image(FILE *file), void __add_trash_data()*.

Основной метод *write_image()* принимает на вход имя файла и пытается по нему открыть файл на запись в бинарном режиме. В случае неудачи выбрасывается исключение. После открытия файла создается структура *png_ptr*, вызывается функция *png_init_io()* и метод *__write_image()*. После выполнения процедуры записи в файл структура *png_ptr* удаляется и файл закрывается.

Метод *__write_image()* получается в качестве аргумента файловый декриптор. Метод записывает в файл заголовок *IHDR*, после этого вызывается метод *__add_trash_data()*, после чего карта пикселей будет записана в файл. В конце в файл также записывается заголовок *IEND*. Если в процессе работы возникнет ошибка, будет выброшено исключение.

Метод *__add_trash_data()* не принимает на вход аргументы. Сперва в методе проверяется наличие уже записанного в байты пикселей выравнивания. В случае, если выравнивание не было записано, производится проверка необходимости добавления муорных данных: в случае, если произведение ширины изображения на величину *bytes_per_pixel* не делится на 4 нацело, необходимо записать в конец каждой строки мусорные данные в количестве от 1 до 3 байтов. В цикле *for* перебираются все строки, под них выделяется новая область памяти, уже с выравниванием, после этого в эту область памяти копируются имеющиеся байты в количестве, определенным выражением: ширина изображения умножить на величину *bytes_per_pixel* и умножить на размер типа *png_byte*. После этого мусорные данные задаются нулями. После данных операций память под исходную строку очищается и указателю на

строку присваивается новая область памяти. В конце работы поле *trd_added*, отвечающее за то, использовалось ли уже выравнивание, получает значение *true.*

2.4 Получение информации о файле

Для получения информации о файле, где записано изображение создан метод класса *PNGImage void get_file_info(const char *filename)*. Данный метод получает в качестве аргумента имя файла. С помощью функции *stat()* из заголовочного файла *sys/stat.h* в структуру *file_info*, объявленную в классе, записывается информация о файле.

3. ОБРАБОТКА ИЗОБРАЖЕНИЙ

Все функции обработки изображений описаны в заголовочном файле *pngimage.h*.

3.1 Вспомогательные функции

Для уменьшения количества повторяющегося кода в процессе написания методов обработки изображения некоторая логика была выделена в отдельные вспомогательные методы. К данным методам относятся методы *_normalize_coords()*, *_set_pixel()*, *_draw_line()*, *_draw_circle()* и *_draw_circle_thickness()*.

Метод *_normalize_coords()* имеет тип возвращаемого значения *void* и принимает 4 аргумента типа указатель на *png_int_32*, отвечающие координатам двух противоположных углов прямоугольной области. Данный метод преобразует полученные координаты в координаты левого верхнего и правого нижнего углов той же прямоугольной области для облегчения вычислений в процессе обработки изображения.

Метод *_set_pixel()* имеет тип возвращаемого значения *void* и принимает 3 аргумента: *png_int_32 x, png_int_32 y*, отвечающие координате пикселя, и *PixelPainted *color*, отвечающий за цвет, в который будет окрашен пиксель. Метод проверяет, не выходит ли заданный пиксель за границы изображения. В случае, если выходит, функция ничего не делает. В противном случае с

помощью оператора *switch* проверяется тип цветовой схемы изображения. Если изображение монохромное без альфа канала, то пиксель окрашивается в среднее арифметическое компонент *r*, *g*, *b* заданного цвета *color*. В случае монохромного изображения с альфа каналом цвет задается аналогичным образом, как и в случае монохромного изображения, однако еще задается компонента *a* равной компоненте *a* заданного цвета *color*. Если изображение имеет цветовую схему *RGB*, компонентам *r*, *g*, *b* пикселя будут заданы значения соответствующих компонент цвета *color*. В случае изображения, имеющего цветовую схему *RGBA*, будут выполнены действия, как и в случае *RGB*, однако компоненте *a* пикселя будет установлено значение компоненты *a* цвета *color*.

Метод *_draw_line()* имеет тип возвращаемого значения *void* и принимает на вход 6 аргументов: 4 аргумента типа *png_int_32*, отвечающие за координаты начала и конца линии, аргумент *int thickness*, отвечающий за толщину линии, и аргумент *PixelPainted *colot*, отвечающий за цвет линии. Алгоритм рисования линии основан на алгоритме Брезенхема для линии с толщиной.

Метод *_draw_circle()* имеет тип возвращаемого значения *void* и принимает на вход 5 аргументов: первые 3 аргумента отвечают за позицию центра окружности и величину ее радиуса, следующий аргумент *int thickness* отвечает за толщину линии окружности, последний аргумент *PixelPainted *color* отвечает за цвет линии окружности. Алгоритм рисования окружности основан на алгоритме Брезенхема для окружности. Однако для реализации толщины линии отрисовывается новая окружность с радиусом *r+thickness-1*, а пространство между окружностями (и есть толщина) заполняется в методе *_draw_circle_thickness()*.

Метод *_draw_circle_thickness()* имеет тип возвращаемого значения *void* и принимает на вход 5 аргументов: первые 3 аргумента отвечают за позицию центра окружности и величину ее радиуса, следующий аргумент *int thickness* отвечает за толщину линии окружности, последний аргумент *PixelPainted *color* отвечает за цвет линии окружности. В двух вложенных циклах проверяются координаты точки, лежащей в четверти квадрата, описанного

около данной окружности. Если точка лежит на месте, где должна будет отрисоваться толщина окружности, то она будет закрашена цветом *color*, также как и 3 других точки, симметричные данной относительно центра окружности.

3.2 Отражение области

Для реализации отражения области относительно оси Ox либо оси Oy был разработан метод *mirror()*, имеющий тип возвращаемого значения *void* и принимающий 5 аргументов: первый аргумент *int axis* отвечает за ось, относительно которой будет отражена область, может быть равен значениям, заданным макросами *mirrorAxisX* и *mirrorAxisY* (макросы заданы в заголовочном файле *pngimage.h*), остальные 4 аргумента имеют тип *png_int_32* и отвечают за координаты прямоугольной области, которая будет отражена. В методе сперва координаты передаются во вспомогательный метод *_normalize_coords()*, после чего производится взаимодействие с областью. В двух вложенных циклах *for* идет перебор половины области изображения и в еще одном вложенном цикле происходит swap байтов пикселей.

3.3 Рисование пентаграммы в круге

Для рисования пентаграммы в круге было разработано два варианта метода *draw_pentagram()*, имеющие тип возвращаемого значения *void*. Первый вариант метода принимает 5 аргументов: координаты центра окружности, радиус окружности, толщину линий и цвет линий. Второй вариант метода принимает 6 аргументов: координаты 2-х противоположных углов прямоугольной области, в которую будет вписана пентаграмма, толщину линий и цвет линий. Вторая реализация метода использует метод *_normalize_coords()*, после чего вычисляет радиус окружности и координаты ее центра и вызывает первый вариант реализации (сделано это для уменьшения количества повторов кода). Первая реализация рисует окружность с координатами центра, радиусом и толщиной с цветом заливки, переданными как аргументы в метод. После этого вычисляются длины вспомогательной линии и длина стороны правильного пятиугольника, вписанного в эту окружность. Также вычисляются отступы, зависящие от длины линии (для красивого отображения). После этого

с помощью вспомогательного метода *_draw_line()* отрисовываются 5 линий, образующих пентаграмму.

3.4 Рисование прямоугольника

Для рисования прямоугольника было разработано два варианта метода *draw_rectangle()*, имеющие тип возвращаемого значения *void*: вариант с заливкой и без нее. Метод без заливки принимает 6 аргументов: 4 аргумента типа *png_int_32*, отвечающие за координаты 2-х противоположных углов прямоугольника, аргумент *int thickness*, отвечающий за толщину линий, и аргумент *PixelPainted *color*, отвечающий за цвет линий. Метод с заливкой дополнительно принимает еще один аргумент: *PixelPainted *fill_color* - , а вместо *color* предпоследний аргумент имеет название *line_color*. Метод без заливки сперва вызывает метод *_normalize_coords()*, после чего отрисовывает все 4 стороны прямоугольника с помощью вспомогательного метода *_draw_line()*. Также одна из линий имеет отступ по *y*, равный половине толщины линий. Метод с заливкой сперва вызывает метод *_normalize_coords()*, после чего в двух циклах *for* закрашивает область с помощью метода *_set_pixel()* цветом *fill_color*. После заливки области прямоугольника отрисовывается сам прямоугольник с помощью метода без заливки.

3.5 Рисование правильного шестиугольника

Для рисования правильного шестиугольника было разработано четыре варианта метода *draw_hexagon()*, имеющие тип возвращаемого значения *void:* 2 варианта с заливкой и 2 без нее. Первый вариант метода принимает на вход 5 аргументов: координаты центра окружности, в которую вписан шестиугольник, ее радиус, толщина линий и цвет линий. Второй вариант метода принимает на вход 6 аргументов: первые 5 такие же, как и в предыдущем варианте, а последний — цвет заливки. Третий вариант метода принимает на вход 6 аргументов: координаты 2-х противоположных углов прямоугольной области, в которую будет вписан шестиугольник, толщину линий и цвет линий. Последний, четвертый вариант метода принимает на вход 7 аргументов: первые 6 такие же, как и в предыдущем варианте, а последний — цвет заливки.

Варианты метода, которые принимают на вход координаты прямоугольной области, вызывают сперва метод *_normalize_coords()*, после чего высчитывают координаты центра окружности, в которую будет вписан шестиугольник, а также ее радиус, после — вызывают варианты метода, принимающие на вход координаты центра окружности и ее радиус. В реализации первого варианта метода вычисляются координаты всех вершин шестиугольника, после чего с помощью вспомогательного метода *_draw_line()* отрисовывается сам шестиугольник. В реализации второго варианта метода сперва закрашивается четвертями область, ограниченная шестиугольником, после чего вызывается первый вариант метода.

4. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПРИЛОЖЕНИЯ

Все файлы графического интерфейса находятся в директории *gui/*. Все иконки, используемые приложением, находятся в директории *icons/*. Графический интерфейс пользователя написан с помощью библиотеки QT 5-й версии.

4.1 Элементы графического интерфейса

Для взаимодействия с пользователем был разработан графический интерфейс, который включает в себя основное окно и некоторые диалоговые окна. Класс *Interface*, описывающий интерфейс приложения, объявлен в заголовочном файле *interface.h*.

Основное окно приложения состоит из нескольких частей: верхнее меню, боковая панель инструментов и рабочая область. Верхнее меню создается в методе *createMenu()* класса *Interface*. Меню содержит список действий «Файл», содержащий пункты *«Открыть»*, *«Сохранить как»* и *«Выход»*. Также меню содержит пункт *«Изображение»*, отвечающий за вывод информации об изображении, и пункт *«О программе»*, отвечающий за вывод справки по приложению. Боковая панель инструментов создается в методе *createTools()* класса *Interface*. Боковая панель содержит кнопки настройки области выделения, кнопки обработки изображения, кнопки настройки инструментов

обработки, в том числе и выбора цвета, и кнопки масштабирования изображения. Рабочая область представляет собой область, где размещается изображение, при большом масштабе появляются полосы прокрутки.

Диалоговые окна, имеющиеся в приложении, можно разделить на 2 типа: информационные и меню настроек. К первому типу относятся такие диалоговые окна, как окно вывода ошибки, создаваемое методом *showErrorMessage()* и содержащее сообщение о возникшей ошибке, окно «О программе», создаваемое методом *About()* и содержащее краткую справку по работе с приложением, окно «Изображение», создаваемое методом *Properties()* и отображающее информацию об открытом изображении (если изображение еще не открыто, все поля будут помечены как «Неизвестно»). Ко второму типу относятся окна, как окно настроек выделения области, создаваемое методом *callSettings()* и содержащее настройки области выделения, окно *«Дополнительные настройки»*, создаваемое методом *callSetAdditionalSettings()* и содержащее настройки заливки рисуемых фигур и настройки отражения области, окно настройки толщины, создаваемое методом *setThickness()* и содержащее настройки толщины линий.

4.2 Обработчики действий пользователя

Для обработки запросов пользователя были разработаны некоторые методы класса *Interface*. Это методы вызова окон открытия и сохранения файла с изображением *OpenFile()* и *SaveFile()*, методы вызова справки и меню *«Изображение» Properties()* и *About()*, метод обработки выхода из приложения *OnQuit()*. Это также методы вызова диалоговых окон с различными настройками *callSettings()*, *callSetAdditionalSettings()*, *setThickness()*, *callSetPrimaryColor()*, *callSetSecondaryColor()*, а также обработчики нажатий по кнопкам обработки изображения: *callDrawPentagram()*, *callMirrorArea()*, *callDrawHexagon()*, *callDrawRectangle()* - и обработчики нажатий по кнопкам увеличения/уменьшения масштаба изображения *UpScale()* и *DownScale()*. После открытия изображения либо его изменения в процессе обработки, изображение необходимо перерисовывать, этим и занимается метод *drawImg()*.

ЗАКЛЮЧЕНИЕ

Была разработана программа, выполняющая поставленные задачи. Данная программа была разделена на подпрограммы, выполняющие задачи считывания, записи и обработки изображения и задачи взаимодействия с пользователем посредством графического интерфейса.

При этом был реализован требуемый функционал: отражение заданной области, рисование пентаграммы в круге, рисование прямоугольника, рисование правильного шестиугольника.

Функционал считывания и записи изображения был реализован при помощи библиотеки *libpng*. Обработка изображения производилась путем изменения карты пикселей.

Программа успешно компилируется и работает на платформе ОС *Linux*.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Справочник по библиотеке *libpng* / libpng.org URL: www.libpng.org/pub/png/libpng-manual.txt

2. Онлайн справочник по C/C++ / cplusplus.com URL: https://cplusplus.com/

3. Описание алгоритмов растеризации прямых, окружностей и т. д. / A Rasterizing Algorithm for Drawing Curves URL: http://members.chello.at/%7Eeasyfilter/Bresenham.pdf

4. Документация по библиотеке *QT* / doc.qt.io URL: https://doc.qt.io/