



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ
ЗАПИСКА
*К КУРСОВОМУ ПРОЕКТУ
НА ТЕМУ:***

Разработка инструмента для построения трёхмерных изображений, ориентированного на микроконтроллеры STM32

Студент группы ИУ7-53Б

(Подпись, дата)

А.М. Сапожков

(И.О. Фамилия)

Руководитель курсовой работы

(Подпись, дата)

Ю.В. Строганов

(И.О. Фамилия)

2022 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ7
(Индекс)

И. В. Рудаков
(И. О. Фамилия)

« ____ » _____ 20 ____ г.

З А Д А Н И Е на выполнение курсового проекта

по дисциплине Компьютерная Графика

Студент группы ИУ7-53Б

Сапожков Андрей Максимович
(Фамилия, имя, отчество)

Тема курсового проекта Разработка инструмента для построения трёхмерных изображений, ориентированного на микроконтроллеры STM32

Направленность КП(учебный, исследовательский, практический, производственный, др.)
учебный

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание: разработать программу, позволяющую пользователю синтезировать трёхмерные изображения при помощи микроконтроллеров семейства STM32. Реализовать интерфейс, который позволит задавать исходные данные в виде файла, содержащего в себе описание моделей, их расположения на сцене и освещения. Исследовать возможности микроконтроллеров семейства STM32 для их применения при решении задач компьютерной графики.

Оформление курсового проекта:

Расчетно-пояснительная записка на 20-40 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку задачи, введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.

Перечень графического (илюстративного) материала (чертежи, плакаты, слайды и т.п.)
На защиту проекта должна быть предоставлена презентация, состоящая из 15-20 слайдов.
На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, интерфейс, результаты проведенных исследований.

Дата выдачи задания « ____ » _____ 20 ____ г.

Руководитель курсового проекта

(Подпись, дата)

Ю. В. Строганов

(И.О. Фамилия)

Студент

(Подпись, дата)

А. М. Сапожков

(И.О. Фамилия)

Содержание

Введение	6
1 Аналитический раздел	7
1.1 Описание платформы STM32	7
1.2 Оценка существующих инструментов	7
1.2.1 Библиотека для построения трёхмерных поверхностных моделей	8
1.2.2 Библиотека для создания анимаций на основе векторной трёхмерной графики	8
1.2.3 Библиотека для создания анимаций трёхмерных каркасных моделей	8
1.2.4 Частичная реализация библиотеки OpenGL для STM32 . . .	8
1.3 Требования к алгоритмам визуализации трёхмерных объектов . .	9
1.4 Анализ алгоритмов удаления невидимых линий и поверхностей . .	10
1.4.1 Алгоритм Робертса	10
1.4.2 Алгоритм Варнока	10
1.4.3 Алгоритм Вейлера-Азертона	11
1.4.4 Алгоритм, использующий Z-буфер	12
1.4.5 Алгоритм построчного сканирования, использующий Z-буфер	13
1.4.6 Алгоритм, использующий список приоритетов	13
1.4.7 Алгоритм определения видимых поверхностей путем трассировки лучей	14
1.5 Анализ алгоритмов закрашивания	15
1.5.1 Простая закраска	15
1.5.2 Закраска по Гуро	15
1.5.3 Закраска по Фонгу	16
1.6 Вывод из аналитического раздела	16
2 Конструкторский раздел	17
2.1 Особенности архитектуры ARM	17
2.2 Описание семейства микроконтроллеров STM32	18
2.3 Структуры данных	18
2.4 Разработка алгоритмов	19
2.4.1 Алгоритм работы программы	19
2.4.2 Алгоритм Варнока	19
2.5 Структура программного комплекса	19
2.6 Вывод из конструкторского раздела	20
3 Технологический раздел	21
3.1 Выбор и обоснование средств разработки	21

3.2	Формат задания исходных данных	21
3.2.1	Задание геометрии трёхмерных моделей	21
3.2.2	Задание характеристик материалов	22
3.2.3	Задание источников света	22
3.3	Структура программы	22
3.4	Детали реализации алгоритмов	24
3.5	Взаимодействие с программой	25
3.5.1	Установка зависимостей	26
3.5.2	Подготовка исходных данных	26
3.5.3	Подключение микроконтроллера	26
3.5.4	Запуск программы	26
3.6	Примеры работы программы	26
3.7	Вывод из технологического раздела	27
4	Экспериментально-исследовательский раздел	28
4.1	Цель проводимых измерений	28
4.2	Описание проводимых измерений	28
4.3	Инструменты измерения времени работы программы	29
4.4	Результаты проведённых измерений	29
4.5	Вывод из экспериментально-исследовательского раздела	31
	Заключение	33
	Список использованных источников	34
A	Функциональная схема визуализации трёхмерной сцены	37
B	Алгоритм Варнока	40
V	Пример исходных данных для работы программы	42
G	Изображения тестовых моделей	44

Глоссарий

Рендеринг — процесс получения изображения по двумерной или трёхмерной модели с помощью компьютерной программы.

Введение

В современном мире компьютерная графика является неотъемлемой частью нашей жизни. Сегодня она используется в различных сферах жизнедеятельности человека, таких как визуализация данных, компьютерные игры, кинематограф и многие другие. Поэтому перед людьми, создающими трехмерные сцены, встает задача создания реалистичных изображений, которые будут учитывать оптические явления преломления, отражения и рассеивания света, свойства поверхностей, тени, а также восприятие окружающего мира человеческим глазом.

Когда встает задача изображения трёхмерной сцены на конкретной вычислительной машине, для каждой платформы нередко приходится разрабатывать отдельные программные решения, учитывающие особенности аппаратного обеспечения. Одной из платформ, которая нуждается в программном обеспечении, решая задачи компьютерной графики, является архитектура ARM и в частности семейство микроконтроллеров STM32. Такая необходимость возникает вследствие роста популярности микроконтроллеров этого семейства по всему миру, в том числе и в России, а также расширения области их промышленного применения. Уже сейчас STM32 используется в отраслях, в которых необходимо решать задачи компьютерной графики, начиная с промышленной автоматики и заканчивая пользовательской электроникой и устройствами интернета вещей.

Цель данной работы – реализовать программный инструмент для построения моделей трехмерных объектов, ориентированный на микроконтроллеры семейства STM32.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

- 1) описать структуру трехмерной сцены, включая объекты, из которых состоит сцена, и определить способ задания исходных данных;
- 2) выбрать и адаптировать существующие алгоритмы трехмерной графики, позволяющие визуализировать трехмерную сцену, для выполнения на заданном оборудовании;
- 3) реализовать выбранные и адаптированные алгоритмы построения трехмерной сцены;
- 4) исследовать возможности микроконтроллеров семейства STM32 для их применения при решении задач компьютерной графики.

1 Аналитический раздел

В данном разделе будут рассмотрены перспективы работы с микроконтроллерами семейства STM32 и существующие решения поставленной задачи, будут проанализированы методы моделирования трёхмерных объектов, среди которых будет выбран метод решения поставленной задачи.

1.1 Описание платформы STM32

Производитель микроконтроллеров семейства STM32, компания STMicroelectronics, является одним из мировых лидеров по производству и продажам микроконтроллеров и прочно закрепилась на российском рынке микроэлектроники. 32-битные микроконтроллеры от этого производителя нашли широкий спектр приложений, разрабатываемых средними и малыми клиентами (так называемый массовый рынок) в рамках промышленных применений [1].

Одной из главных причин популярности микроконтроллеров семейства STM32 является наличие обширной экосистемы, которая развивается с течением времени. К микроконтроллерам семейства STM32 можно подключать всё больше периферийных устройств для решения различных задач. Также экосистема STM32 имеет программную поддержку в виде бесплатной среды разработки и интегрируется с программными инструментами сторонних компаний, что даёт большую гибкость при разработке программных продуктов для STM32.

Стоит отметить, что рассматриваемое семейство микроконтроллеров имеет ряд аналогов, в том числе и от российских производителей, благодаря чему имеется возможность в будущем перенести разработки с STM32 на отечественную аппаратную платформу.

1.2 Оценка существующих инструментов

Поиск по популярному сервису для хостинга ИТ проектов GitHub [2] показал, что уже существуют четыре программных инструмента с открытым исходным кодом, решающих поставленную задачу. Их оценку можно проводить по следующим критериям:

- 1) производительность;
- 2) универсальность;
- 3) качество пользовательского интерфейса.

Рассмотрим существующие программные решения.

1.2.1 Библиотека для построения трёхмерных поверхностных моделей

Библиотека, представленная в [3], предоставляет функционал для построения трёхмерных моделей и рассчитан на конкретные дисплеи. Визуализация осуществляется с помощью алгоритма построчного сканирования, использующего Z-буфер и простую модель освещения с учётом плоского затенения. Важно заметить, что в коде библиотеки не используются низкоуровневые оптимизации. Исходные данные для работы алгоритмов задаются в коде программы, то есть отсутствует возможность загрузки модели из файла или с помощью графического пользовательского интерфейса.

1.2.2 Библиотека для создания анимаций на основе векторной трёхмерной графики

Библиотека, представленная в [4], предназначена для рендеринга трёхмерных анимаций с частотой кадров до 80 кадров в секунду. Однако этот инструмент предназначен для работы с конкретными моделями микроконтроллера и дисплея, обладающего относительно низким разрешением — 128x160 пикселей. Как и в предыдущем случае, исходные данные задаются в коде программы.

1.2.3 Библиотека для создания анимаций трёхмерных каркасных моделей

Функционал библиотеки, представленной в [5], позволяет осуществлять рендеринг трёхмерных анимаций на конкретной модели дисплея с использованием низкоуровневых оптимизаций. Но стоит отметить, что при изображении трёхмерной модели не используются алгоритмы удаления невидимых линий и поверхностей, что очень сильно упрощает данный инструмент. Как и в предыдущих примерах, исходные данные задаются в коде программы.

1.2.4 Частичная реализация библиотеки OpenGL для STM32

Программный инструмент, представленный в [6], рассчитан на конкретные модели микроконтроллеров и реализует базовый функционал популярной графической библиотеки OpenGL, а именно построение трёхмерных моделей с помощью алгоритма удаления невидимых линий и поверхностей, использующего Z-буфер. В алгоритмах не учитывается модель освещения, тени, текстурирование и сглаживание. Несмотря на сложность используемых алгоритмов, инструмент обладает высоким быстродействием за счёт наличия низкоуровневых оптимизаций. Данная библиотека, как и предыдущие, не предусматривает взаимодействия с пользователем и строит трёхмерные модели, заданные в коде программы.

После анализа библиотек можно выделить недостатки, встречающиеся у существующих программных инструментов:

- 1) ориентированность исключительно на конкретные модели микроконтроллеров и дисплеев;
- 2) относительно низкая производительность за счёт отсутствия низкоуровневых оптимизаций;
- 3) способ задания исходных данных затрудняет использование графического инструмента;
- 4) отсутствует пользовательский интерфейс;
- 5) отсутствует масштабируемость.

Обладая перечисленными выше недостатками, программный инструмент не может использоваться в полномасштабных проектах.

Также стоит отметить, что популярные графические библиотеки, такие как OpenGL [7], Direct3D [8] или Vulkan [9], которые повсеместно используются при решении задач компьютерной графики на различных аппаратных платформах, не поддерживают работу с микроконтроллерами семейства STM32.

Отсюда следует необходимость разработки программного инструмента, который решал бы поставленную задачу для платформы STM32, не имея при этом недостатков, присущих рассмотренным ранее альтернативам.

1.3 Требования к алгоритмам визуализации трёхмерных объектов

При выборе алгоритма удаления невидимых линий и поверхностей необходимо учитывать особенности решаемой задачи:

- 1) микроконтроллеры имеют относительно небольшую вычислительную мощность;
- 2) объём доступной памяти микроконтроллеров очень ограничен и составляет, как правило, не более 0.5–2 Мб.

Исходя из этих особенностей, можно сформулировать требования к алгоритмам:

- 1) компактность – алгоритмы должны задействовать минимальное количество памяти, используя структуры данных, не содержащие избыточной информации;
- 2) простота и лаконичность – алгоритмы должны быть простыми, чтобы итоговый исполняемый файл занимал как можно меньше места в памяти микроконтроллера;
- 3) быстродействие – микроконтроллер, не обладающий большими вычислительными мощностями, должен выполнять алгоритмы за приемлемое время.

Ввиду того, что в данном случае ограничение по памяти сильнее ограничения по времени, эффективность алгоритмов по памяти будет самым важным критерием оценки алгоритмов.

Рассмотрим основные алгоритмы построения трехмерной сцены [10].

1.4 Анализ алгоритмов удаления невидимых линий и поверхностей

1.4.1 Алгоритм Робертса

Алгоритм Робертса – метод удаления невидимых линий и поверхностей, работающий в объектном пространстве. Основные этапы работы алгоритма Робертса:

0) подготовка исходных данных: разбиение невыпуклых тел на выпуклые, со-ставление корректной матрицы каждого тела;

Далее для каждого тела сцены:

- 1) удаление рёбер, экранируемых самим телом;
- 2) удаление рёбер, экранируемых другими телами (если на сцене находятся несколько тел);
- 3) удаление линий пересечения тел, экранируемых самими телами и другими телами, связанными отношением протыкания.

Математические методы, используемые в этих этапах, дают очень высокую точность вычислений за счёт того, что алгоритм Робертса работает в объектном пространстве.

В то же время, каждый этап алгоритма основан на трудоёмких математических вычислениях, сложность которых растёт как квадрат числа объектов сцены. Стоит отметить, что алгоритм Робертса можно оптимизировать, однако реализовать эти оптимизации крайне сложно.

1.4.2 Алгоритм Варнока

Алгоритм Варнока решает задачу удаления невидимых линий и поверхностей в пространстве изображения. Его идея состоит в том, что на обработку областей изображения, содержащих мало информации, должно тратиться очень мало времени, и наоборот.

В пространстве изображения рассматривается окно и решается вопрос о том, достаточно ли оно простое для визуализации. Если это не так, то окно разбивается на подокна до тех пор, пока условие не будет выполняться.

В конкретных реализациях могут использоваться различные методы разбиения окна и рассматриваться различные случаи взаимного расположения многоуголь-

ников относительно окна. Усложнение критериев и способов разбиения способно значительно повысить эффективность алгоритма.

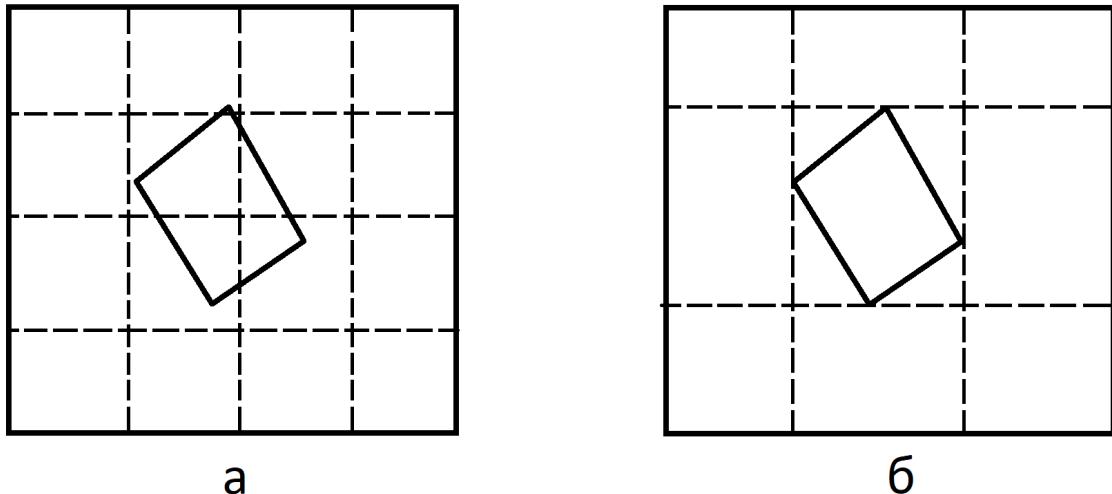


Рисунок 1.1 — Сравнение двух способов разбиения окна

Также в целях оптимизации можно выполнять предварительную сортировку многоугольников по глубине с целью оптимизации поиска многоугольников, охватывающих рассматриваемое окно.

Стоит отметить, что рекурсивное разбиение окна, в зависимости от числа пересечений объектов, может стать как положительной, так и отрицательной стороной алгоритма Варнока. Чем меньше пересечений объектов сцены, тем быстрее выполнится алгоритм.

1.4.3 Алгоритм Вейлера-Азертона

В алгоритме Вейлера-Азертона производится попытка минимизировать количество шагов в алгоритме Варнока путем разбиения окна вдоль границ многоугольника.

Алгоритм решения задачи удаления невидимых линий и поверхностей удаления целиком базируется на алгоритме отсечения тех же авторов. Работа будет вестись с проекциями многоугольников, то есть в пространстве изображения. Алгоритм отсечения позволяет найти как внутренние, так и внешние отсечения, что важно.

Основные этапы алгоритма Вейлера-Азертона:

- 1) сортировка многоугольников по глубине;
- 2) отсечение всех многоугольников сцены по границе ближайшего к наблюдателю многоугольника;

- 3) удаление многоугольников, экранируемых ближайшим к наблюдателю многоугольником;
- 4) рекурсивное разбиение многоугольников, выполняемое в том случае, когда многоугольники, расположенные в списке за ближайшим многоугольником.

К недостаткам алгоритма можно отнести то, что он не справляется с ситуацией пересечения двух многоугольников, поэтому в нём приходится отдельно рассматривать этот случай и разбивать один из них по линии пересечения многоугольников.

Ввиду больших затрат времени и памяти на выполнение алгоритма отсечения Вейлера-Азертона, основанного на работе с двунаправленными циклическими списками, соответствующий алгоритм удаления невидимых линий и поверхностей нельзя считать более эффективным, чем алгоритм Варнока.

1.4.4 Алгоритм, использующий Z-буфер

Алгоритм, использующий Z-буфер является одним из простейших алгоритмов удаления невидимых поверхностей, который работает в пространстве изображения.

Для своей работы он использует два буфера: буфер кадра, хранящий цвет (интенсивность) каждого пикселя в пространстве изображения, и Z-буфер (буфер глубины), хранящий информацию о координате Z для каждого пикселя. Вначале в Z-буфер заносятся минимальные значения Z , а буфер кадра заполняется фоновыми значениями. Затем каждый многоугольник преобразуется в растровую форму и записывается в буфер кадра, при этом не производится начального упорядочения.

В процессе работы глубина (значение координаты Z) каждого нового пикселя, который надо занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесён в Z-буфер. Если это сравнение показывает, что новый пиксель расположен ближе к наблюдателю, чем пиксель, уже находящийся в буфере кадра, то новый пиксель заносится в буфер кадра. Кроме того, производится корректировка Z-буфера: в него заносится глубина нового пикселя. Если же глубина рассматриваемого пикселя меньше глубины пикселя, хранящегося в Z-буфере, то никаких действий производить не надо.

Достоинства:

- простота визуализации сцен любой сложности;
- отсутствие сортировки объектов сцены по глубине, как в других алгоритмах.

Недостатки:

- большой объём задействуемой памяти;
- трудоёмкость устранения лестничного эффекта;

- трудоёмкость реализации эффектов, связанных с полупрозрачностью просвечиванием и рядом других специальных задач, повышающих реалистичность изображения.

Последний недостаток связан с тем, что пиксели заносятся в буфер кадра в произвольном порядке. Это затрудняет получение информации, необходимой для методов, основывающихся на предварительном анализе изображения. Проблема относительно легко решается использованием методов постфильтрации.

Ввиду того, что алгоритм требует объём памяти, в разы превышающий тот, который имеют микроконтроллеры семейства STM32, его рассмотрение для реализации поставленной задачи не имеет смысла.

1.4.5 Алгоритм построчного сканирования, использующий Z-буфер

Использование методов построчного сканирования в алгоритме, использующем Z-буфер, решается проблема использования большого количества памяти за счёт того, что буферизуется не экран, а только одна сканирующая строка.

Алгоритмы построчного сканирования работают в пространстве изображения и обрабатывают сцену в порядке прохождения сканирующей строки. Для каждого пикселя вычисляется глубина многоугольника, пересекающего рассматриваемую сканирующую строку.

Количество вычислений можно сократить, если использовать понятие интервалов. Решение задачи удаления невидимых поверхностей сводится к выбору видимых отрезков в каждом интервале, полученном путём деления сканирующей строки проекциями точек пересечения рёбер.

За то, что алгоритм построчного сканирования, использующий Z-буфер, решает проблему большого расхода памяти, приходится расплачиваться дополнительными вычислениями, связанными с использованием списка активных многоугольников.

1.4.6 Алгоритм, использующий список приоритетов

В основе алгоритма лежит сортировка объектов по приоритету, то есть по глубине объектов сцены или их расстоянию от точки наблюдения. Сначала изображаются объекты, расположенные дальше всех от наблюдателя, а затем их перекрывают объекты, находящиеся ближе к наблюдателю.

После сортировки объектов сцены по глубине и получается первоначальный список приоритетов. Далее он корректируется путём выполнения серии тестов на экранирование для каждой пары многоугольников в списке. Эти тесты достаточно трудоёмкие с точки зрения эффективности по времени и сложности реализации.

Также стоит отметить, что в алгоритме очень сложно идентифицируются случаи пересечения и циклического перекрытия многоугольников сцены.

1.4.7 Алгоритм определения видимых поверхностей путем трассировки лучей

В алгоритме, использующем трассировку лучей, отслеживаются (трассируются) лучи, идущие от наблюдателя к объекту. Такая трассировка называется обратной.

Для определения видимых поверхностей от наблюдателя, находящегося в бесконечности на положительном направлении оси Z, испускаются лучи, проходящие через каждый пиксель картинной плоскости. Траектория каждого луча отслеживается, чтобы определить пересечения объектов сцены с данным лучом. Пересечение с максимальным значением z представляет видимую поверхность для данного пикселя картинной плоскости.

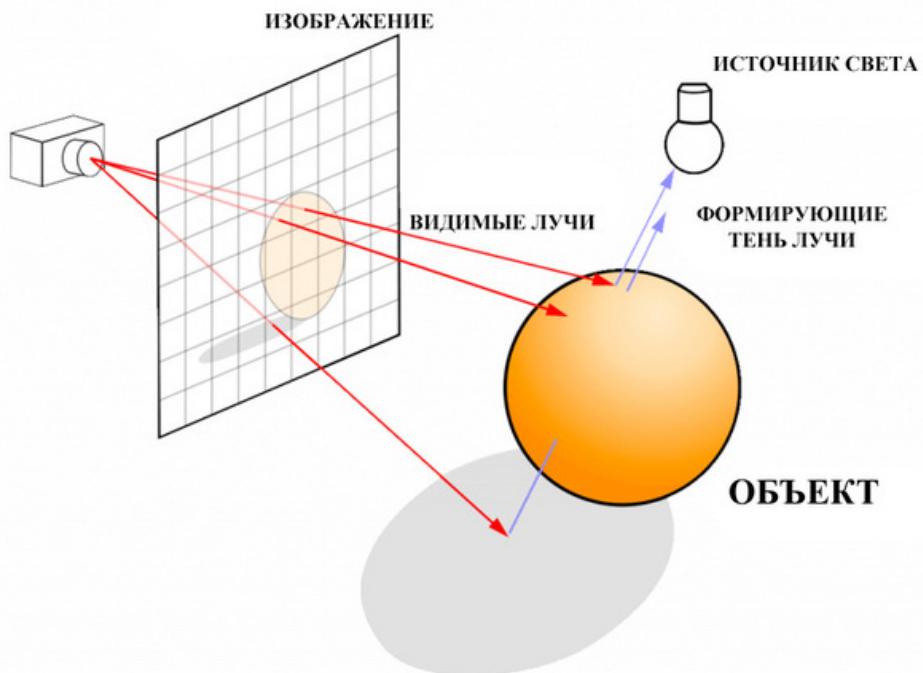


Рисунок 1.2 — Иллюстрация алгоритма определения видимых поверхностей путем трассировки лучей

Поиск пересечений является наиболее важным и трудоемким элементом этого алгоритма, поэтому его быстродействие существенно зависит от скорости их определения. Однако, несмотря на трудоёмкость этой процедуры, вычислительная сложность метода линейно зависит от количества объектов сцены.

Главное преимущество алгоритма трассировки лучей — качество и реалистичность изображения. Также при использовании данного алгоритма нетрудно реализовать наложение света и тени на объекты.

К недостаткам данного алгоритма можно отнести производительность, так как трассировка большого количества лучей является очень трудоёмким процессом. Учёт прозрачности, усложняющий процесс трассировки лучей, только усугубит проблему.

1.5 Анализ алгоритмов закрашивания

1.5.1 Простая закраска

При использовании простой закраски вся грань многогранника закрашивается одним уровнем интенсивности, который вычисляется по закону Ламберта.

Предпосылки к использованию простой закраски:

- 1) источник света находится на бесконечном удалении от объектов сцены;
- 2) наблюдатель находится на бесконечном удалении от объектов сцены;
- 3) закрашиваемая грань реально существует, а не является результатом аппроксимации другой поверхности.

Из последнего пункта вытекает недостаток алгоритма простой закраски: он не подходит для закраски криволинейных поверхностей. Также к недостаткам алгоритма можно отнести то, что он никак не учитывает отражённый свет.

Главное преимущество алгоритма - простота реализации. Если поместить наблюдателя в бесконечность, то данный алгоритм хорошо подойдёт для решения поставленной задачи.

1.5.2 Закраска по Гуро

Закраска по Гуро предполагает билинейную интерполяцию интенсивностей. В результате вычисления интенсивности в каждой точке грани создаётся иллюзия гладкой криволинейной поверхности.

Закраска по Гуро хорошо сочетается с простой моделью освещения с диффузным отражением.

Несмотря на то, что применение закраски по Гуро сильно улучшает качество изображения, этот алгоритм имеет свои недостатки:

- 1) появляется эффект полос Маха;
- 2) в результате сглаживания теряется граница между гранями многогранника, что в некоторых случаях может дать неверный результат.

Последний недостаток можно устраниТЬ, разбивая грани многогранника или создавая возмущения в уравнения нормалей к граням, чтобы сделать их разными. Однако такой подход увеличит объём вычислений.

1.5.3 Закраска по Фонгу

Закраска по Фонгу предполагает билинейную интерполяцию нормалей к граням многогранника. Такой подход требует больших вычислительных затрат, чем закраска по Гуро, и в результате даёт более качественное изображение, устранивая большинство недостатков предыдущего алгоритма.

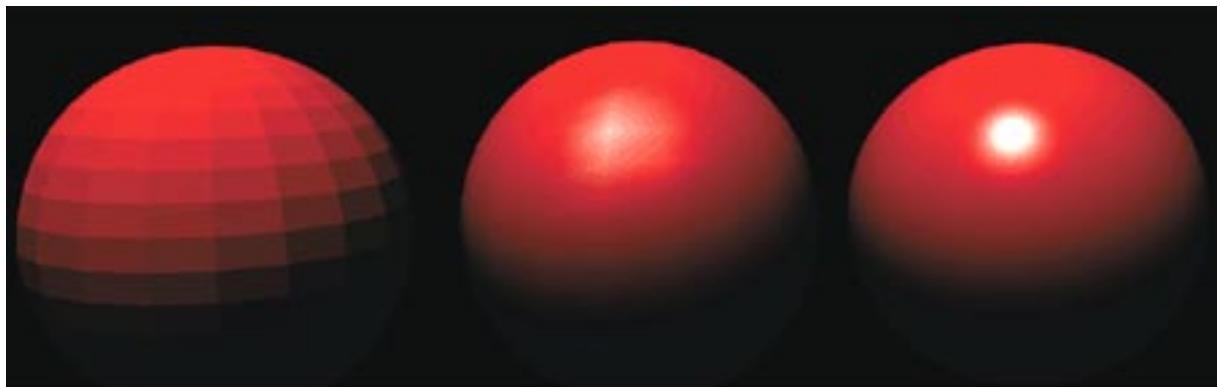


Рисунок 1.3 — Сравнение методов закраски: слева — простая, в центре — Гуро, справа — Фонга

К недостаткам закраски по Гуро и по Фонгу можно отнести ошибки при закраске невыпуклых многоугольников, а также большую вычислительную сложность и ресурсоёмкость.

1.6 Вывод из аналитического раздела

В данном разделе были проанализированы перспективы работы с семейством STM32, существующие решения поставленной задачи, а также методы моделирования трёхмерных объектов. Ввиду отсутствия универсальных программных инструментов для платформы STM32, которые могли бы решать задачи трёхмерной компьютерной графики, а также с учётом переносимости программного обеспечения на отечественные платформы, разработка имеет актуальность.

Несмотря на перспективы импортозамещения, для разработки была выбрана оригинальная платформа STM32 ввиду её более развитой экосистемы и наличия большого количества материалов по работе с микроконтроллерами этого семейства. Это упростит работу с микроконтроллером и его периферийными устройствами, а также разработку программных продуктов для данной платформы, что позволит уделить больше внимания задачам компьютерной графики.

В качестве алгоритма удаления невидимых линий и поверхностей был выбран алгоритм Варнока ввиду его компактности и относительно небольшой вычислительной сложности, а в качестве алгоритма закраски - простая закраска по причине её относительно небольшой ресурсоёмкости и вычислительной сложности.

2 Конструкторский раздел

В данном разделе будут описаны особенности аппаратной платформы STM32, а также алгоритмы и структуры данных, выбранные для решения поставленной задачи, будет разработана структура программного комплекса.

2.1 Особенности архитектуры ARM

Архитектура ARM — система команд и семейство описаний и готовых топологий 32-битных и 64-битных микропроцессорных/микроконтроллерных ядер.

Архитектура развивалась с течением времени и, начиная с ARMv7, были определены три профиля: для устройств, требующих высокой производительности (смартфоны, планшеты), для приложений, работающих в реальном времени, и для микроконтроллеров и бюджетных встраиваемых устройств [11].

Основные отличия архитектуры ARM от x86, которая используется в современных ПК: [12]

- использование упрощенного набора инструкций - **RISC** (Reduced Instruction Set Computing);
- предикация - возможность условного исполнения практически любой команды;
- упрощённая работа с памятью за счёт использования единого адресного пространства для всех устройств вычислительной машины;
- более объёмная регистровая архитектура [13] [14];
- меньшее энергопотребление;
- системы, базирующиеся на процессорах ARM, проще масштабировать и отлаживать.

Архитектура ARM поддерживается множеством операционных систем, к числу которых относятся Linux (в том числе Android, основанная на ядре ОС Linux), iOS, BSD, macOS Big Sur. Также на платформе запускаются отдельные варианты семейства ОС Windows: Windows CE, Windows Phone, Windows RT, Windows 10.

Процессоры ARM широко используются в потребительской электронике, к которой относятся смартфоны, плееры, портативные игровые консоли, калькуляторы, умные часы и компьютерные периферийные устройства. Эти процессоры имеют низкое энергопотребление, поэтому находят широкое применение во встраиваемых системах и преобладают на рынке мобильных устройств, для которых данный фактор критически важен [15].

2.2 Описание семейства микроконтроллеров STM32

Многие лицензиаты готовых топологий ядер ARM проектируют собственные топологии ядер на базе системы команд ARM. Одной из таких компаний является STMicroelectronics, производящая различные полупроводниковые электронные и микроэлектронные компоненты, к которым относятся микроконтроллеры семейства STM32.

STM32 — семейство микроконтроллеров, основанных на 32-битных ядрах ARM Cortex-M. Каждый микроконтроллер состоит из ядра процессора, статической RAM-памяти, флеш-памяти, отладочного и различных периферийных интерфейсов.

Дизайн ядра ARM имеет множество настраиваемых опций, и для каждого микроконтроллера выбирается индивидуальная конфигурация с добавлением своих собственных периферийных устройств к ядру микроконтроллера перед преобразованием дизайна в полупроводниковую пластину.

Основные преимущества микроконтроллеров STM32:

- низкая стоимость;
- гибкая и масштабируемая экосистема;
- большой выбор сред разработки;
- высокая производительность;
- наличие инструментов для отладки микроконтроллера.

Самым важным преимуществом STM32 является взаимозаменяемость чипов, которая достигается за счёт универсальности ядра STM32, позволяющего менять производителя с минимальными затратами на программный код. Также внутри семейства STM32 поддерживается pin-to-pin совместимость, которая позволяет менять объем памяти (флэш-память и ОЗУ) и периферию (Ethernet, USB, CAN и так далее), не меняя печатную плату. Таким образом, если для решения какой-либо задачи не хватает ресурсов одного микроконтроллера, его можно заменить на более мощный, не меняя самой схемы и платы [16].

2.3 Структуры данных

Чтобы формализовать алгоритм синтеза изображения в программе, необходимо определить структуры данных, которые будут в ней использоваться. В данной работе приняты следующие соглашения:

- 1) трёхмерные модели являются полигональными, тогда сцену можно представить в виде массива многоугольников (полигонов);
- 2) многоугольник включает в себя следующие данные:
 - количество вершин;
 - массив x и у координат вершин;

- коэффициенты уравнения поверхности, несущей данный многоугольник, заданного в виде $a * x + b * y + c * z = 1000$;
 - цвет;
- 3) окна, использующиеся в алгоритме Варнока, имеют прямоугольную форму и хранят следующую информацию:
- количество многоугольников, рассматриваемых при изображении данного окна;
 - массив многоугольников;
 - координаты x и y левой нижней и правой верхней вершин окна.

2.4 Разработка алгоритмов

2.4.1 Алгоритм работы программы

Диаграмма, оформленная в соответствии с нотацией IDEF0 и отражающая общую декомпозицию алгоритма работы программы, представлена в приложении А.

2.4.2 Алгоритм Варнока

Для удаления невидимых линий и поверхностей был выбран алгоритм Варнока, основанный на рекурсивном разбиении окон. Под окном в данном алгоритме понимается область изображения на дисплее, которая может содержать визуализируемые объекты сцены. Разбиение окон в алгоритме Варнока может быть реализовано как рекурсивно, так и итерационно. В данной работе будет рассматриваться итерационная реализация с использованием стека окон, так как она задействует меньший объём памяти, чем рекурсивная. Максимальная длина стека окон составляет

$$] \log_2(\max(w; h)) [+ 2 * (] \log_2(\min(w; h)) [+ 1) + 2, \quad (2.1)$$

где w – горизонтальное разрешение дисплея, h – вертикальное разрешение дисплея.

Схема алгоритма Варнока представлена в приложении Б на рисунке Б.1 На рисунке Б.2 представлен алгоритм идентификации взаимного расположения окна и рассматриваемого многоугольника. Алгоритм следует оформить в виде подпрограммы.

2.5 Структура программного комплекса

На рисунке 2.1 представлена структура программного комплекса, оформленная в виде диаграммы развёртывания. Она отражает компоненты, необходимые для работы системы, а также способы их взаимодействия.

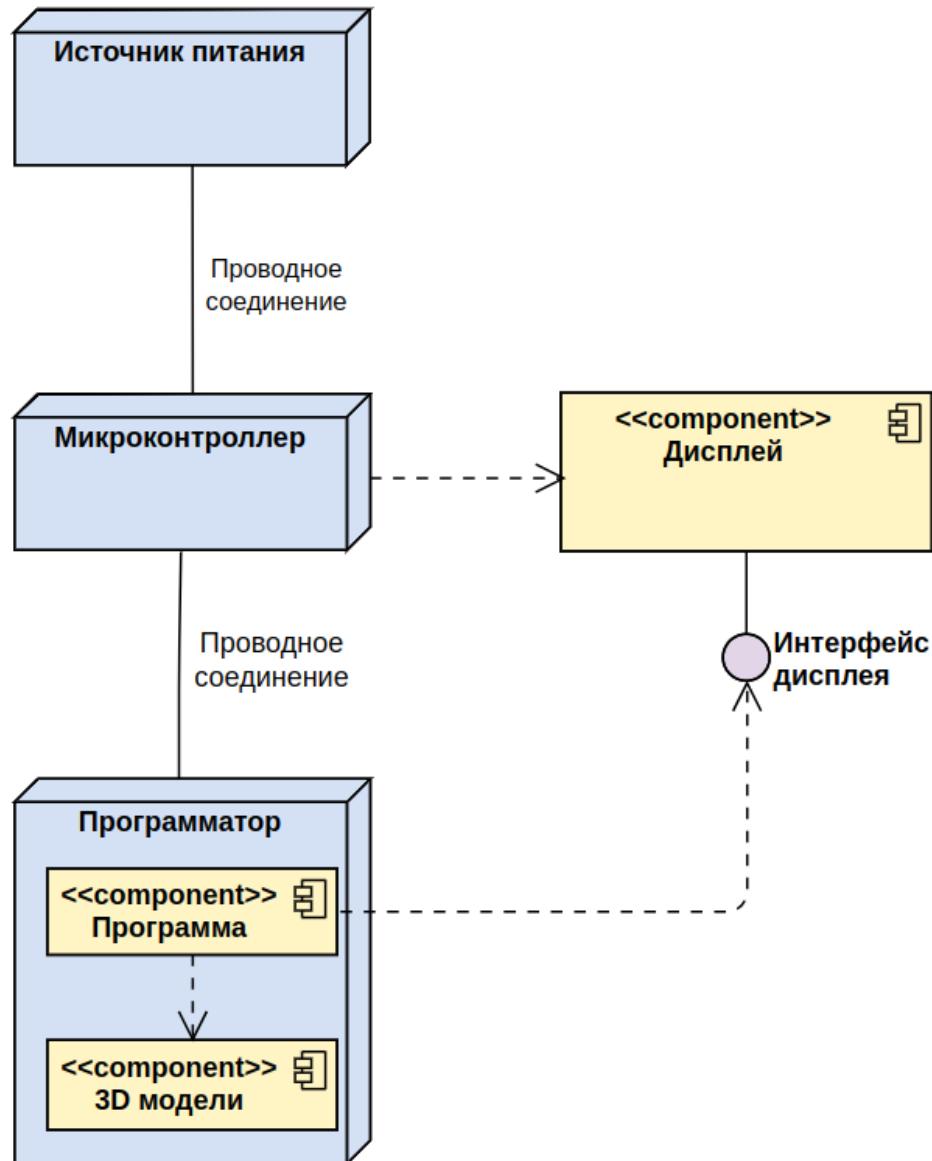


Рисунок 2.1 — Диаграмма развёртывания программного комплекса

2.6 Вывод из конструкторского раздела

В данном разделе были описаны особенности работы с аппаратной платформой STM32, структура программного комплекса, а также алгоритмы и структуры данных, выбранные для решения поставленной задачи и отвечающие требованиям компактности, простоты и быстродействия.

3 Технологический раздел

В данном разделе будут рассмотрены детали реализации программного комплекса, описанного в конструкторской части работы, и приведены примеры работы программы.

3.1 Выбор и обоснование средств разработки

В качестве языка программирования, на котором будет реализовано программное обеспечение, выбран язык C++ [17]. Данный выбор обусловлен следующими причинами:

- 1) высокая скорость работы языка;
- 2) строгая типизация;
- 3) наличие библиотек для работы с микроконтроллерами семейства STM32 и периферийными устройствами;
- 4) наличие большого количества материалов о разработке программного обеспечения для микроконтроллеров семейства STM32 на данном языке;
- 5) возможность сертификации программ, написанных на данном языке [18].

В качестве среды разработки был выбран текстовый редактор Visual Studio Code [19], обладающий большим количеством плагинов и инструментов для создания программного обеспечения на различных языках, в том числе на C++.

Для контроля качества кода использовался статический анализатор кода cppcheck [20] и отладчик использования памяти valgrind [21].

3.2 Формат задания исходных данных

Разработанный графический инструмент использует полигональное представление трёхмерных моделей. Для их задания необходима информация о полигонах, из которых состоит модель, а именно их геометрическое строение и цвет. Также для построения трёхмерной сцены необходима информация о расположении источников света.

3.2.1 Задание геометрии трёхмерных моделей

В качестве формата, задающего геометрическое строение и расположение полигонов на сцене, был выбран формат obj [22]. Это простой формат данных, который представляет только трехмерную геометрию, а именно координаты вершин, индексы вершин, составляющие полигоны, а также текстуры и нормали. Важно заметить, что с форматом obj могут работать различные редакторы трёхмерной графики. Самыми популярными среди них являются Blender [23], Autodesk AutoCAD [24], Adobe

Photoshop [25] и КОМПАС-3D [26]. Пример файла формата obj приведён в приложении В (листиング В.1).

3.2.2 Задание характеристик материалов

Формат obj помимо геометрии задаёт материалы, из которых выполнены заданные полигоны. Эти материалы подробно описывает формат mtl [27], который содержит в себе информацию о цвете и оптических свойствах полигонов, из которых состоит трёхмерная модель. Пример файла формата mtl приведён в приложении В (листиинг В.2).

3.2.3 Задание источников света

Форматы obj и mtl описывают только трёхмерные модели не предусматривают задания расположения источников света на сцене. Поэтому для представления этих данных был разработан формат lgt, в котором построчно задаются вещественные координаты x, y, z источников света. Пример файла формата lgt приведён в приложении В (листиинг В.3).

3.3 Структура программы

В программе реализованы следующие модули:

reader — модуль чтения исходных данных из файлов;

converter — модуль подготовки исходных данных для алгоритмов визуализации;

writer — модуль записи сконвертированных данных в промежуточный файл;

display — модуль, предоставляющий данные о конкретном дисплее и функции для работы с ним;

errors — модуль обработки ошибок;

debug — отладочный модуль;

warnock — модуль, реализующий алгоритм Варнока удаления невидимых линий и поверхностей;

wstack — модуль, предоставляющий структуры данных "окно", "стек окон" и операции над ними;

data_structures — модуль, предоставляющий структуры данных, описывающие полигональные модели.

Диаграмма модулей программы представлена на рисунке 3.1.

Программа разделена на две части. Первая предназначена для предобработки исходных данных и выполняется на компьютере. Она получает на вход файлы с

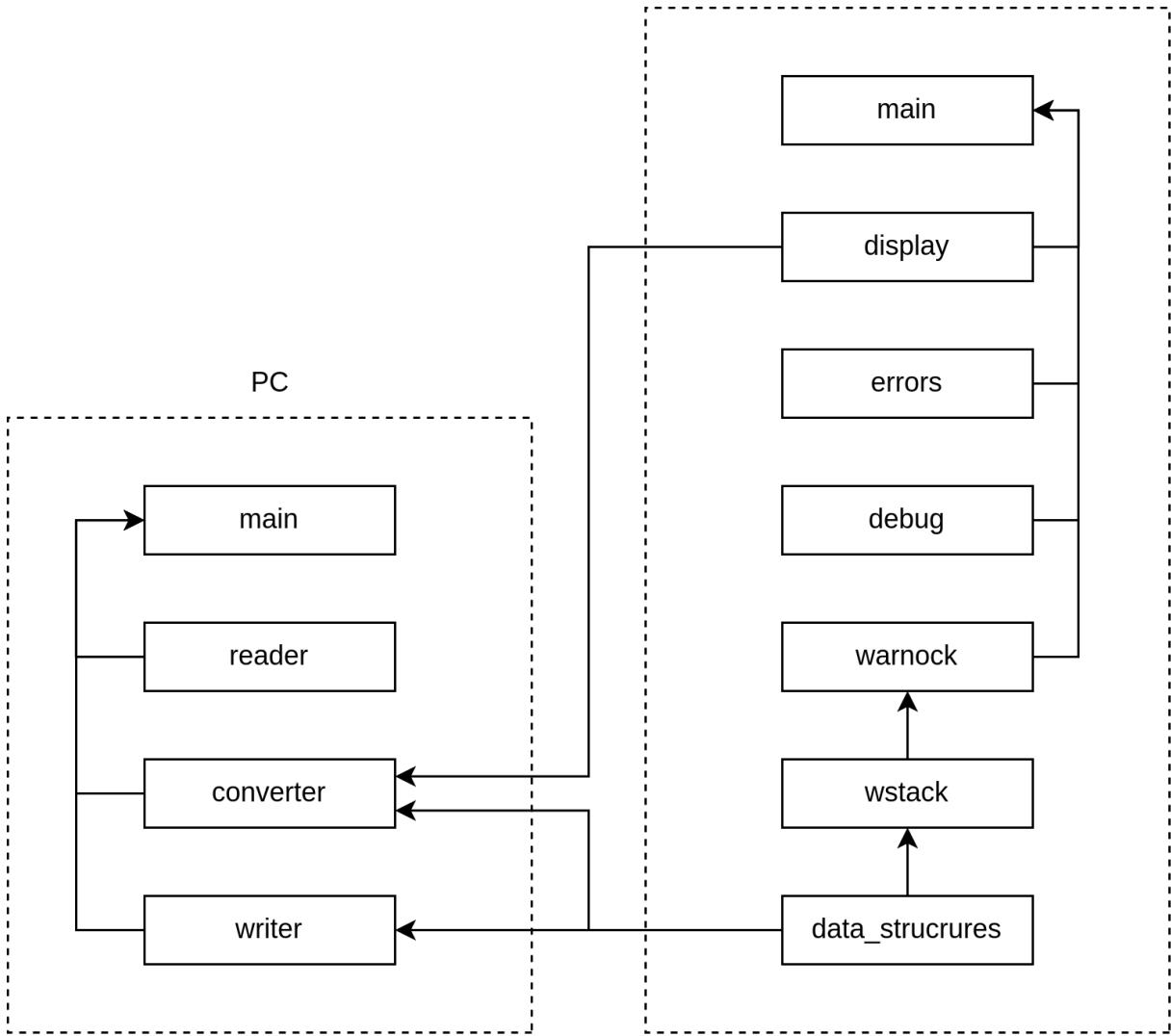


Рисунок 3.1 — Диаграмма модулей программы

исходными данными и транслирует их в файл, который компонуется со второй частью программы, отвечающей за рендеринг изображения. Получившаяся программа загружается на микроконтроллер и выводит на дисплей заданное изображение.

Такая гибкая архитектура позволяет заменять аппаратное обеспечение, на котором должна выполняться программа, не затрагивая при этом алгоритмы рендеринга трёхмерной сцены. Это означает, что данное программное обеспечение обладает достаточно большой универсальностью с точки зрения требований к аппаратному обеспечению и не зависит от конкретных моделей микроконтроллера и дисплея.

Также для развертывания и запуска программы был написан сценарий сборки, предназначенный для запуска на операционных системах семейства Linux [28].

3.4 Детали реализации алгоритмов

В листинге 3.1 представлен код функции warnock_subdivide, которая реализует алгоритм Варнока удаления невидимых линий и поверхностей.

Листинг 3.1 — Алгоритм Варнока

```
1 errors_t warnock_subdivide(const window_t &window, const uint16_t bgcolor,
2   void (*set_pixel)(uint16_t, uint16_t, uint16_t))
3 {
4   wstack_t stack = wstack_init();
5
6   if (!wstack_allocate(stack, window))
7   {
8     wstack_free(stack);
9
10  return MEMORY_ALLOCATE_ERROR;
11 }
12
13 polygon_t nearest_polygon;
14
15 while (stack.size > 0)
16 {
17   window_t wcur;
18   wstack_pop(stack, wcur);
19   uint32_t end_outside = 0;
20   uint32_t start_encomparassing = wcur.polygons.count;
21   uint32_t i = 0;
22
23   while (i < start_encomparassing)
24   {
25     polygon_t cur_polygon = wcur.polygons.array[i];
26     int overlap_flag = overlapping(cur_polygon, wcur);
27
28     if (overlap_flag == OUTSIDE)
29       swap(wcur.polygons.array[i++],
30             wcur.polygons.array[end_outside++]);
31     else if (overlap_flag == ENCOMPARASSING)
32       swap(wcur.polygons.array[i],
33             wcur.polygons.array[--start_encomparassing]);
34     else
35       ++i;
36   }
37
38   polygons_t new_arr = {wcur.polygons.count - end_outside,
39   wcur.polygons.array + end_outside};
```

```

37     if ((wcur.pend.x - wcur.pbeg.x == 1) and (wcur.pend.y -
38         wcur.pbeg.y == 1))
39         fill_pixel(wcur.pbeg, new_arr, bgcolor, set_pixel);
40     else if (start_encomparassing != end_outside)
41         split_window(stack, wcur, new_arr);
42     else
43     {
44         if (start_encomparassing == wcur.polygons.count)
45             fill_window(wcur, bgcolor, set_pixel);
46         else if (nearest_polygon_exists(nearest_polygon, wcur,
47             new_arr))
48             fill_window(wcur, nearest_polygon.color, set_pixel);
49         else
50             split_window(stack, wcur, new_arr);
51     }
52     wstack_free(stack);
53
54     return SUCCEEDED;
55 }
```

Перед анализом каждого окна выполняется идентификация многоугольников с целью их разделения на три группы: внешние, охватывающие и все остальные. В том случае, если необходимо разбиение окна, оно делится на четыре равные части, если это возможно. Перед разбиением из массива многоугольников удаляются внешние, так как они заведомо будут внешними относительно подокон и не будут играть никакой роли при изображении сцены. Данная оптимизация повышает быстродействие алгоритма.

3.5 Взаимодействие с программой

Для программы был разработан консольный интерфейс, а не графический. Такое решение было принято по причине выбора формата задания исходных данных, не требующего от пользователя большого количества действий. Также консольный интерфейс позволяет использовать разработанный программный инструмент другими программами. В будущем это позволит разрабатывать программное обеспечение для семейства микроконтроллеров STM32, использующее трёхмерную графику.

Для запуска программы необходимо выполнить четыре шага: установить зависимости, подготовить исходные данные, подключить микроконтроллер к компьютеру и запустить рендеринг.

3.5.1 Установка зависимостей

Ввиду того, что для изменения выводимого на дисплей изображения программу приходится перекомпоновывать, на компьютере, с которого будет загружаться программа, должны быть установлены инструменты, необходимые для компоновки программы [29]. Также для запуска сценария сборки требуется утилита make [30].

3.5.2 Подготовка исходных данных

Рассмотрим данный и последующие этапы на примере модели куба. Для задания трёхмерных моделей необходимо создать 3 файла. Содержимое файлов cube.obj, cube.mtl и cube.lgt приведено в приложении В. Первые два файла можно экспортировать из графических редакторов, а третий создать с помощью текстового редактора. Полученные файлы необходимо поместить в директорию проекта models/название_модели (в примере директория будет называться models/cube).

3.5.3 Подключение микроконтроллера

Для загрузки программы на микроконтроллер необходимо подключить его к компьютеру с помощью проводного соединения и запомнить абсолютный путь к файлу устройства.

3.5.4 Запуск программы

Для запуска сборки необходимо выполнить в командной оболочке команду вида

```
sudo make view model=cube device=/dev/sdb target=/run/media/user/unit
```

где model - название модели, device - файл подключенного устройства, target - точка монтирования устройства. Далее произойдёт компиляция программы, после чего она загрузится на микроконтроллер и выведет изображение на дисплей.

3.6 Примеры работы программы

На рисунке 3.2 приведено изображение модели куба, рассматриваемое в качестве примера. Другие изображения моделей приведены в приложении Г.

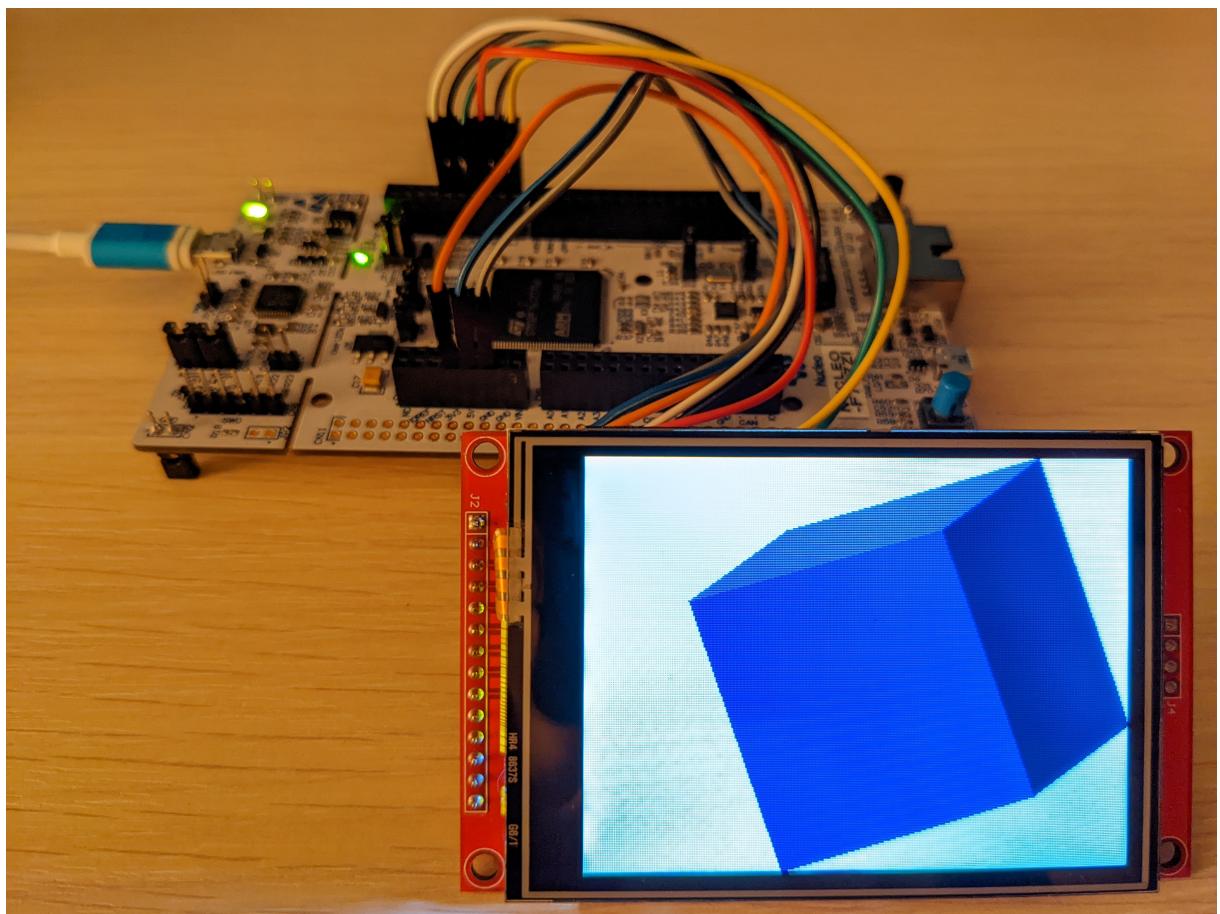


Рисунок 3.2 — Изображение модели куба на дисплее с разрешением 320x240 пикселей

3.7 Вывод из технологического раздела

В данном разделе были рассмотрены детали реализации программного инструмента, разработанного на основе алгоритма Варнока удаления невидимых линий и поверхностей. Также был рассмотрен алгоритм взаимодействия с программой и были приведены примеры её работы.

4 Экспериментально-исследовательский раздел

В данном разделе будет произведено исследование разработанного программного обеспечения с целью определения возможностей применения микроконтроллеров семейства STM32 для решения задач компьютерной графики.

4.1 Цель проводимых измерений

Для программы, которая должна выполняться на микроконтроллере, обладающем сравнительно небольшой вычислительной мощностью, критически важен фактор быстродействия. Особенно он важен при изображении сцены, содержащей большое количество объектов. В связи с этим очень важно понимать, как оборудование, исполняющее разработанную программу, будет справляться с увеличением нагрузки и как при этом будет изменяться ресурсоёмкость реализованных алгоритмов.

Целью проводимых измерений будет проведение нагружочного тестирования и установление зависимости между числом объектов на сцене и затратами времени и памяти на работу алгоритма визуализации.

4.2 Описание проводимых измерений

Для установления зависимости между числом объектов на сцене и временем работы алгоритма будут произведены замеры времени при визуализации трёх моделей, содержащих 312, 1172 и 6228 полигонов. Изображения моделей приведены в приложении Г. В самом начале будет произведён замер времени визуализации небольшой части модели, а затем к ней постепенно будут добавляться новые полигоны до тех пор, пока число изображаемых полигонов не достигнет числа полигонов во всей модели и она не будет полностью изображена. Время будет замеряться в тактах процессора. Время высвечивания пикселей на дисплее учитываться не будет.

Замеры объёма памяти, необходимого для работы программы, будут произведены аналогично замерам времени, то есть путём исследования процесса визуализации отдельных фрагментов изображения. В процессе визуализации каждого фрагмента будет фиксироваться максимальный объём затраченной оперативной памяти. В этот объём входят данные об изображаемых моделях, а также статическая и динамическая память, используемая программой в процессе выполнения. Объём скомпилированного кода программы учитываться не будет, так как он хранится не в оперативной, а в постоянной памяти микроконтроллера.

Для проведения измерений будет использоваться модель микроконтроллера STM32F767ZI [31], обладающая следующими техническими характеристиками:

- модель процессора - Arm Cortex-M7 [32];

- тактовая частота процессора - 216 МГц;
- объём флеш-памяти - 2 Мб;
- объём оперативной памяти - 512 Кб.

Для просмотра получившихся изображений будет использоваться TFT-дисплей с разрешением 320x240 пикселей на базе чипа ILI9341 [33].

4.3 Инструменты измерения времени работы программы

Для измерения реального времени процессора, затраченного на выполнение алгоритма визуализации трёхмерной сцены, будет использоваться отладочный модуль DWT [34]. Данный модуль предоставляет функционал для профилирования работы процессора, в частности для замера количества тактов процессора, прошедших с момента начала работы программы.

4.4 Результаты проведённых измерений

Результаты измерений времени представлены на рисунке 4.1.

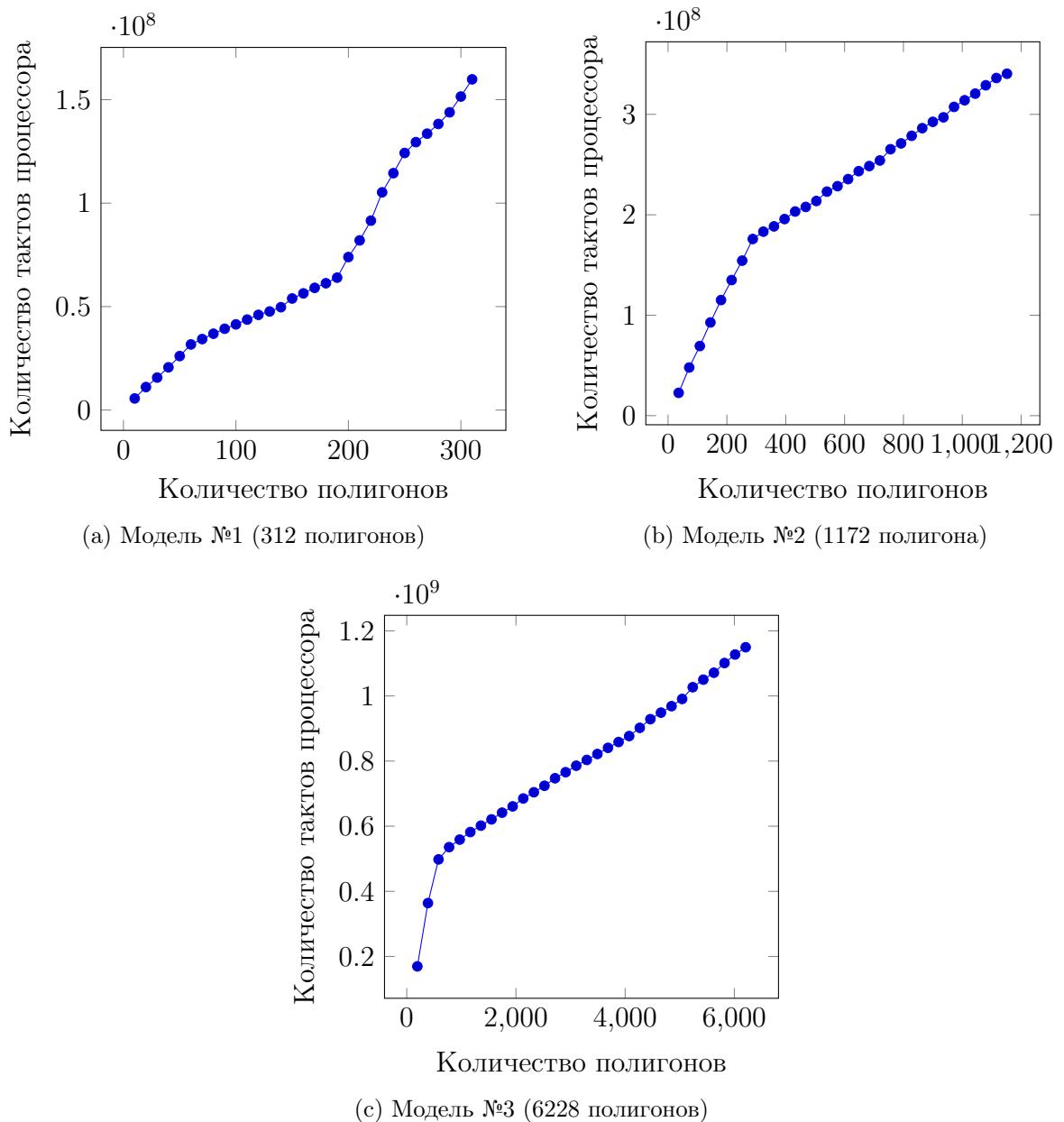


Рисунок 4.1 — Затраты времени на выполнение алгоритма визуализации

Результаты измерений объёма памяти, необходимого программе, представлены на рисунке 4.2.

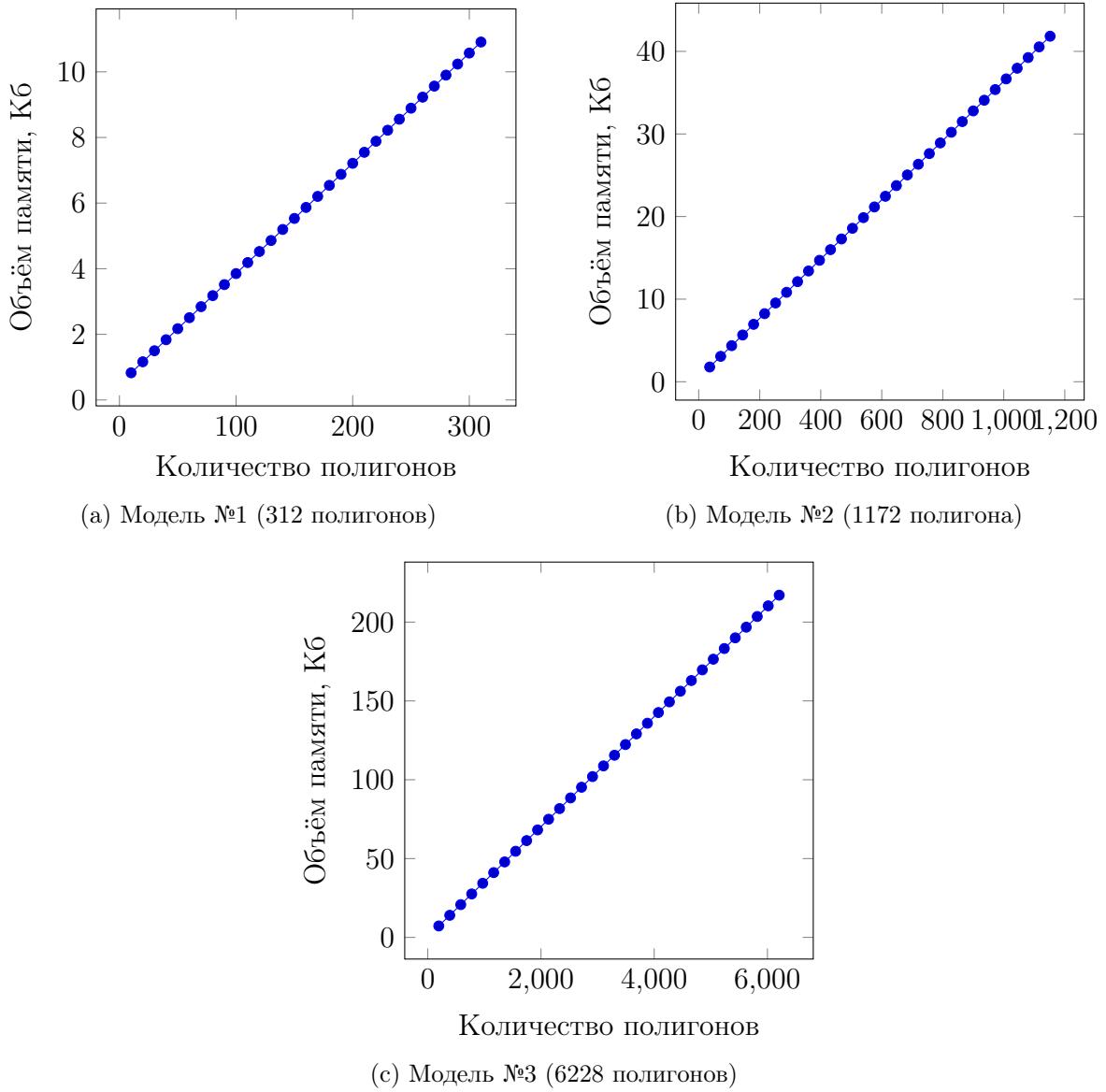


Рисунок 4.2 — Затраты памяти на выполнение алгоритма визуализации

4.5 Вывод из экспериментально-исследовательского раздела

В данном разделе было проведено исследование зависимости между числом объектов на сцене и ресурсоёмкостью алгоритма визуализации.

Результаты измерений показали, что зависимость затрат памяти на работу алгоритма от числа объектов на сцене является линейной. А зависимость затрат времени, предположительно, является кусочно-линейной. Коэффициент пропорциональности между числом объектов на сцене и временем работы алгоритма не является одинаковым на различных участках графиков, так как различные случаи взаимного расположения полигонов обрабатываются с разной скоростью. Увеличение времени работы алгоритма при добавлении на сцену очередного полигона зависит от того, будет ли он пересекаться с полигонами, которые уже находятся на сцене.

Также проведённые измерения показали, что оборудование, исполняющее разработанную программу, равномерно справляется с большой нагрузкой, так как зависимость между числом объектов на сцене и ресурсоёмкостью алгоритма является линейной на всех участках графиков. По результатам тестов, максимальное количество объектов (полигонов), которое микроконтроллер семейства STM32 позволяет обработать, превышает 6000.

Заключение

В рамках курсового проекта был реализован программный инструмент для микроконтроллеров семейства STM32, который позволяет визуализировать трёхмерные модели.

Были рассмотрены существующие алгоритмы удаления невидимых линий и поверхностей и алгоритмы закраски, проанализированы их достоинства, недостатки и возможность использования для решения поставленной задачи. С учётом её особенностей были разработаны структуры данных для реализации выбранных алгоритмов.

Разработанная программа позволяет получать на экране дисплея изображение полигональной модели, заданной пользователем. При разработке были учтены недостатки существующих программных решений для аппаратной платформы STM32.

В ходе выполнения экспериментальной части работы было установлено, что разработанное программное обеспечение на тестовом оборудовании показывает высокую стабильность. При возрастании нагрузки на систему сохраняются эффективность работы программы и качество получаемого результата. Микроконтроллеры семейства STM32 подходят для решения относительно простых задач компьютерной графики, не требующих построения реалистических изображений с определением проекционных теней, использованием глобальной модели освещения и методов закраски, требующих больших вычислительных затрат.

Графический инструмент, разработанный в рамках курсового проекта, имеет по меньшей мере два направления дальнейшего развития.

1. Разработка новых приложений, использующих разработанный графический инструмент. К этому располагают интерфейс программы и формат задания исходных данных.
2. Перенос разработок с STM32 на отечественные аппаратные платформы. Это возможно благодаря наличию аналогов семейства микроконтроллеров STM32 отечественного производства.

Список использованных источников

1. STMicroelectronics: перспективы [Электронный ресурс]. — Режим доступа: <https://www.compel.ru/lib/123541> (дата обращения: 29.07.2022).
2. GitHub [Электронный ресурс]. — Режим доступа: <https://github.com> (дата обращения: 29.07.2022).
3. Tiny3D-engine-STM32 [Электронный ресурс]. — Режим доступа: <https://github.com/Nand-e/Tiny3D-engine-STM32> (дата обращения: 29.07.2022).
4. ST7735-3d-filled-vector [Электронный ресурс]. — Режим доступа: https://github.com/cbm80amiga/ST7735_3d_filled_vector (дата обращения: 29.07.2022).
5. STM32-3D-TEST [Электронный ресурс]. — Режим доступа: https://github.com/LedKast/STM32_3D_TEST (дата обращения: 29.07.2022).
6. cOpenGL [Электронный ресурс]. — Режим доступа: <https://github.com/dizcza/cOpenGL> (дата обращения: 29.07.2022).
7. OpenGL: Related toolkits and APIs [Электронный ресурс]. — Режим доступа: https://www.khronos.org/opengl/wiki/Related_toolkits_and_APIs (дата обращения: 29.07.2022).
8. Direct3D 11 Graphics [Электронный ресурс]. — Режим доступа: <https://docs.microsoft.com/en-us/windows/win32/direct3d11/atoc-dx-graphics-direct3d-11> (дата обращения: 29.07.2022).
9. Vulkan [Электронный ресурс]. — Режим доступа: <https://www.vulkan.org> (дата обращения: 29.07.2022).
10. Роджерс Д.Ф. Алгоритмические основы машинной графики. — Мир, 1989. — ISBN: 9785030004761. — Режим доступа: <https://books.google.ru/books?id=iptuAAAACAAJ>.
11. ARM developer documentation [Электронный ресурс]. — Режим доступа: <https://developer.arm.com/documentation> (дата обращения: 01.08.2022).
12. ARM vs x86 - Explained [Электронный ресурс]. — Режим доступа: <https://www.section.io/engineering-education/arm-x86/> (дата обращения: 01.08.2022).
13. ARM registers [Электронный ресурс]. — Режим доступа: <https://developer.arm.com/documentation/dui0473/m/overview-of-the-arm-architecture/arm-registers> (дата обращения: 01.08.2022).
14. Intel 64 and IA-32 Architectures Software Developer Manuals [Электронный ресурс]. — Режим доступа: <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html> (дата обращения: 01.08.2022).

15. ARM Introduction [Электронный ресурс]. — Режим доступа: <https://www.electronicshub.org/arm-introduction/#Introduction> (дата обращения: 01.08.2022).
16. STM32: эпоха 32-битных микроконтроллеров наступила [Электронный ресурс]. — Режим доступа: <https://www.compel.ru/lib/53953> (дата обращения: 01.08.2022).
17. Standard C++ [Электронный ресурс]. — Режим доступа: <https://isocpp.org/> (дата обращения: 12.08.2022).
18. Документы по сертификации средств защиты информации и аттестации объектов информатизации по требованиям безопасности информации [Электронный ресурс]. — Режим доступа: <https://fstec.ru/normotvorcheskaya/sertifikatsiya> (дата обращения: 23.08.2022).
19. Visual Studio Code - Code Editing. Redefined [Электронный ресурс]. — Режим доступа: <https://code.visualstudio.com/> (дата обращения: 12.08.2022).
20. Cppcheck - A tool for static C/C++ code analysis [Электронный ресурс]. — Режим доступа: <https://cppcheck.sourceforge.io/> (дата обращения: 12.08.2022).
21. Valgrind Home [Электронный ресурс]. — Режим доступа: <https://valgrind.org/> (дата обращения: 12.08.2022).
22. Wavefront OBJ File Format Summary [Электронный ресурс]. — Режим доступа: <https://www.fileformat.info/format/wavefrontobj/egff.htm> (дата обращения: 12.08.2022).
23. Blender [Электронный ресурс]. — Режим доступа: <https://www.blender.org/> (дата обращения: 12.08.2022).
24. AutoCAD: 2D and 3D CAD software [Электронный ресурс]. — Режим доступа: <https://www.autodesk.com/products/autocad/overview> (дата обращения: 12.08.2022).
25. Adobe Photoshop [Электронный ресурс]. — Режим доступа: <https://www.adobe.com/ru/products/photoshop.html> (дата обращения: 12.08.2022).
26. КОМПАС-3D [Электронный ресурс]. — Режим доступа: <https://kompas.ru/kompas-3d/about> (дата обращения: 23.08.2022).
27. WaveFront Material (.mtl) File Format [Электронный ресурс]. — Режим доступа: <https://www.fileformat.info/format/material/> (дата обращения: 12.08.2022).
28. Linux [Электронный ресурс]. — Режим доступа: <https://www.linux.org/> (дата обращения: 12.08.2022).
29. Arm GNU Toolchain [Электронный ресурс]. — Режим доступа: <https://developer.arm.com/Tools%20and%20Software/GNU%20Toolchain> (дата обращения: 12.08.2022).

30. GNU Make [Электронный ресурс]. — Режим доступа: <https://www.gnu.org/software/make/> (дата обращения: 12.08.2022).
31. STM32F767ZI [Электронный ресурс]. — Режим доступа: <https://www.st.com/en/microcontrollers-microprocessors/stm32f767zi.html> (дата обращения: 17.08.2022).
32. Cortex-M7 [Электронный ресурс]. — Режим доступа: <https://developer.arm.com/Processors/Cortex-M7> (дата обращения: 17.08.2022).
33. ILI9341 Driver Datasheet [Электронный ресурс]. — Режим доступа: <https://datasheetspdf.com/datasheet/ILI9341.html> (дата обращения: 17.08.2022).
34. Data Watchpoint and Trace Unit [Электронный ресурс]. — Режим доступа: <https://developer.arm.com/documentation/ddi0489/f/data-watchpoint-and-trace-unit> (дата обращения: 03.09.2022).

Приложение А Функциональная схема визуализации трёхмерной сцены

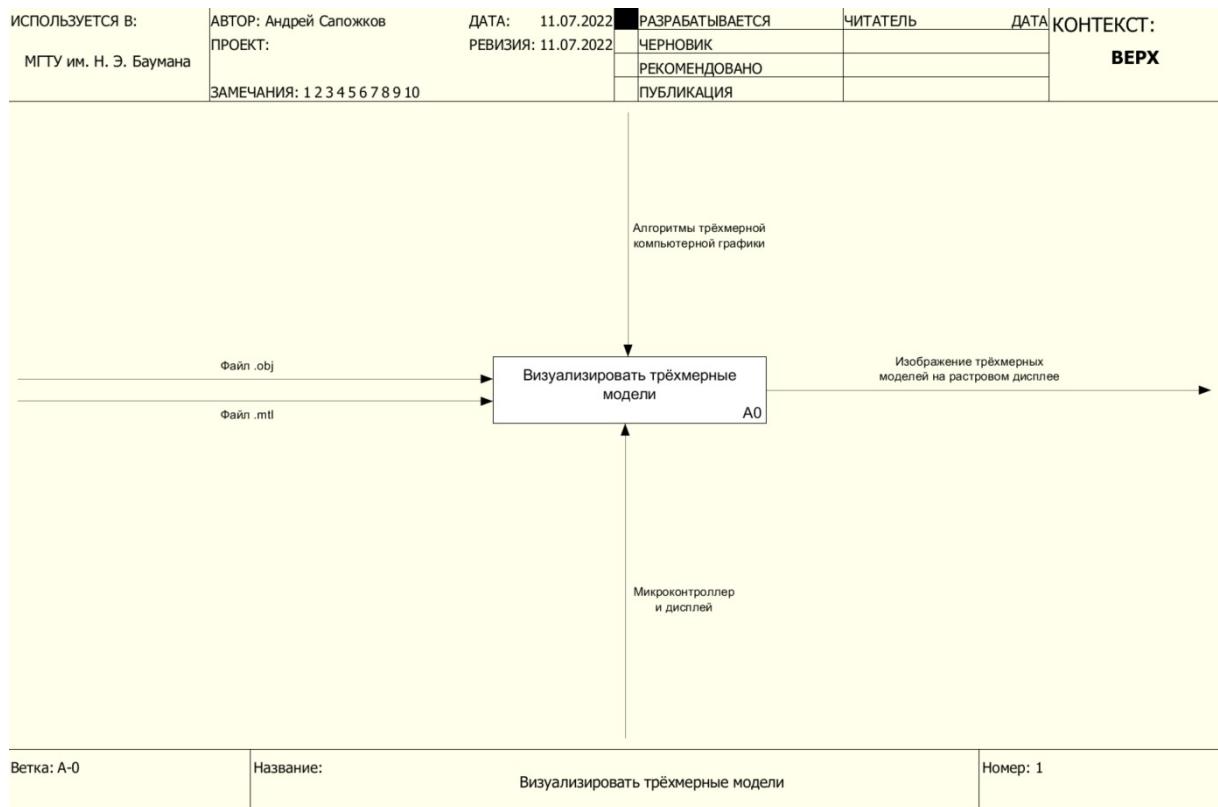


Рисунок А.1 — Функциональная схема визуализации трёхмерной сцены, верхний уровень

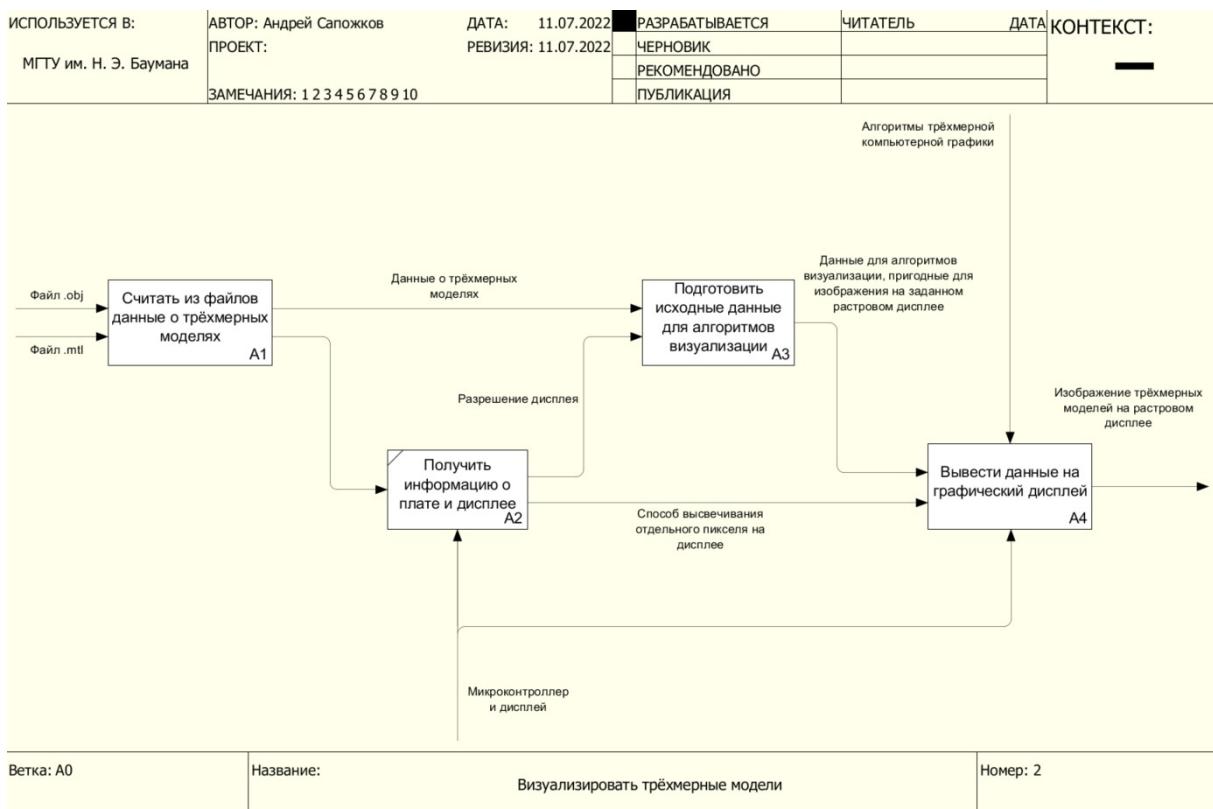


Рисунок А.2 — Функциональная схема визуализации трёхмерной сцены, декомпозиция уровня А0

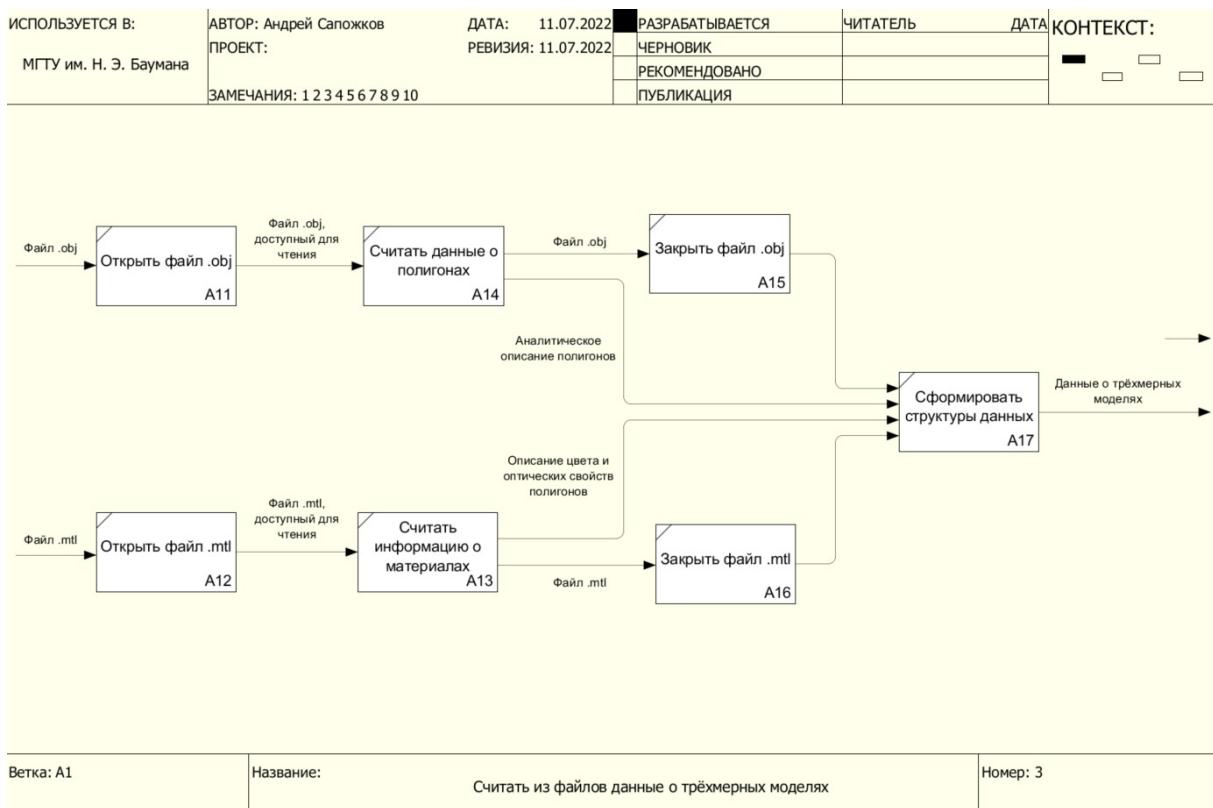


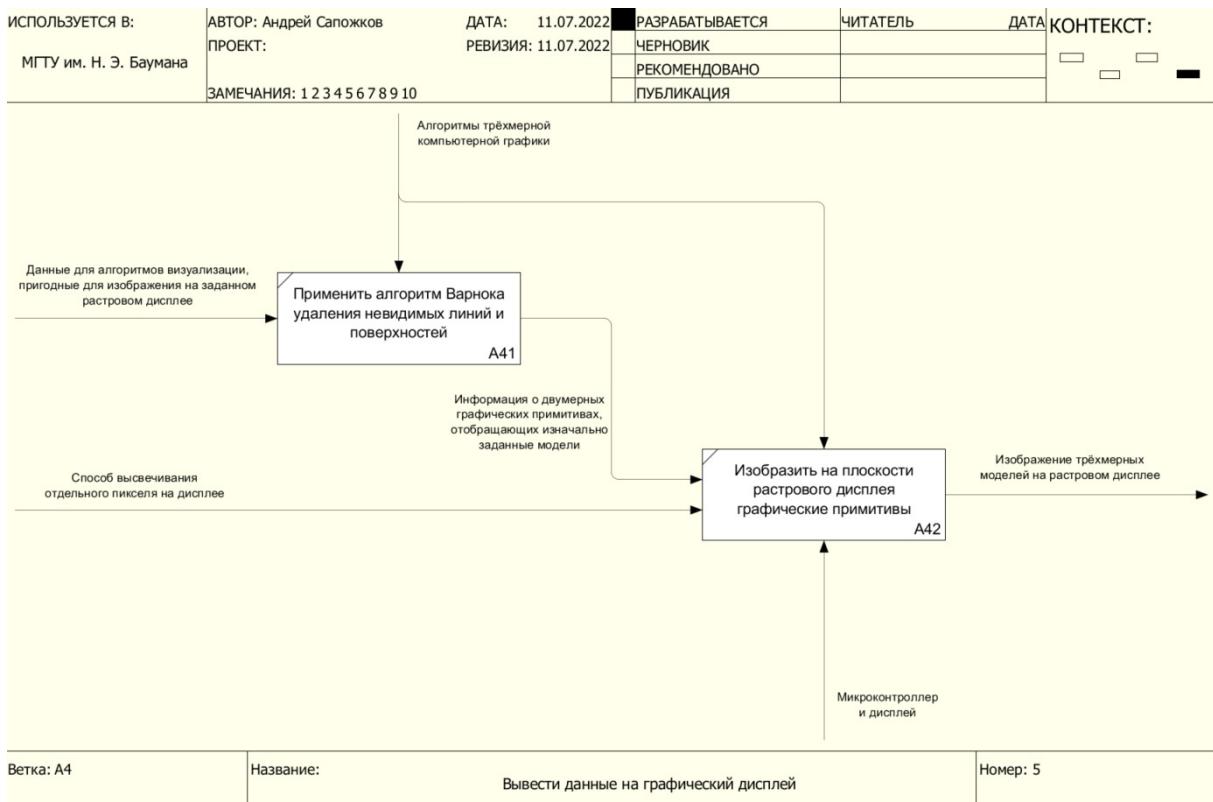
Рисунок А.3 — Функциональная схема визуализации трёхмерной сцены, декомпозиция уровня А1

ИСПОЛЬЗУЕТСЯ В:	АВТОР: Андрей Сапожков	ДАТА: 11.07.2022	РАЗРАБАТЫВАЕТСЯ	ЧИТАТЕЛЬ	ДАТА	КОНТЕКСТ:
МГТУ им. Н. Э. Баумана	ПРОЕКТ:	РЕВИЗИЯ: 11.07.2022	ЧЕРНОВИК			
ЗАМЕЧАНИЯ: 1 2 3 4 5 6 7 8 9 10			РЕКОМЕНДОВАНО			



Ветка: А3	Название:	Подготовить исходные данные для алгоритмов визуализации	Номер: 4
-----------	-----------	---	----------

Рисунок А.4 — Функциональная схема визуализации трёхмерной сцены, декомпозиция уровня А3



Ветка: А4	Название:	Вывести данные на графический дисплей	Номер: 5
-----------	-----------	---------------------------------------	----------

Рисунок А.5 — Функциональная схема визуализации трёхмерной сцены, декомпозиция уровня А4

Приложение Б Алгоритм Варнока

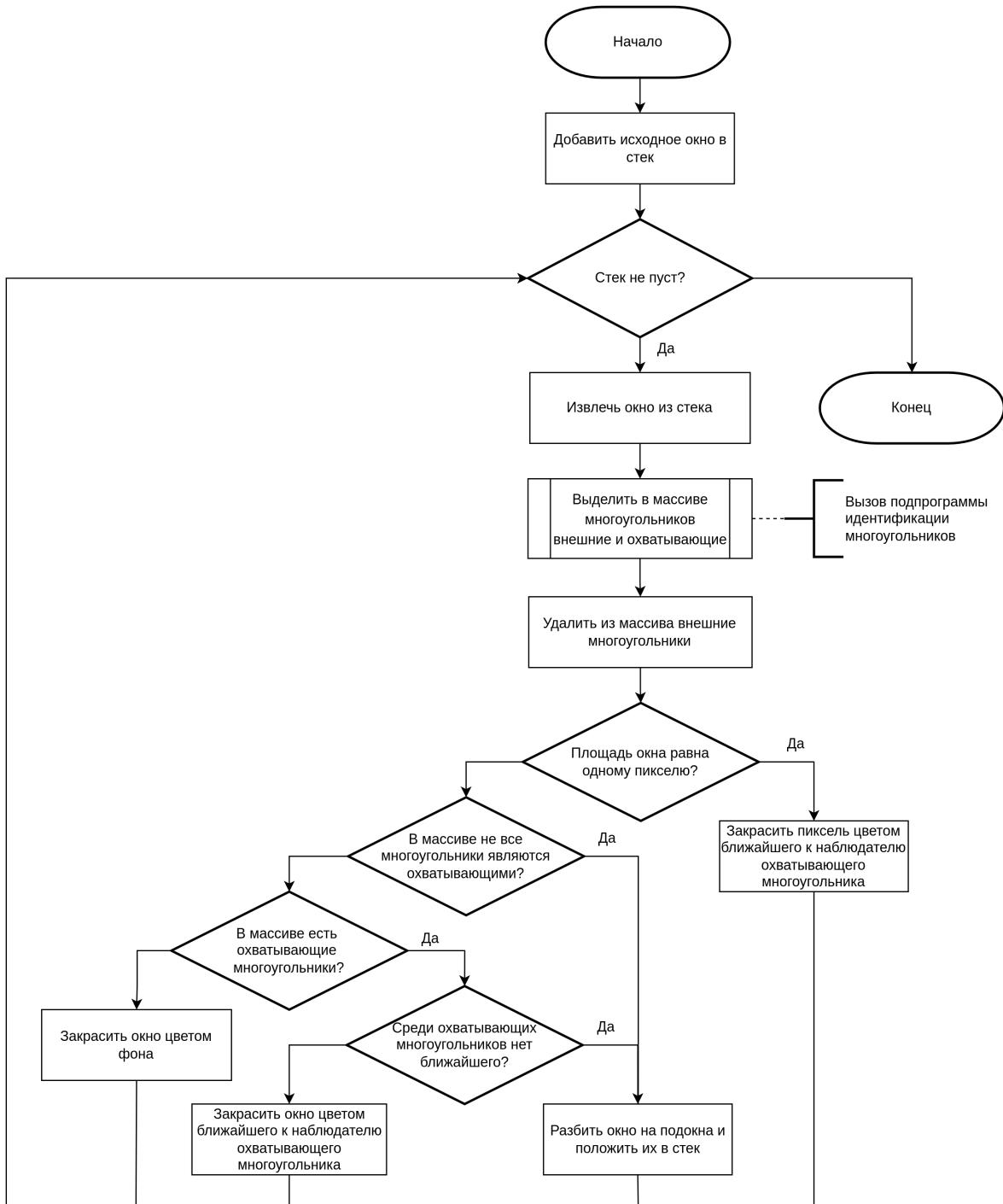


Рисунок Б.1 — Схема алгоритма Варнока

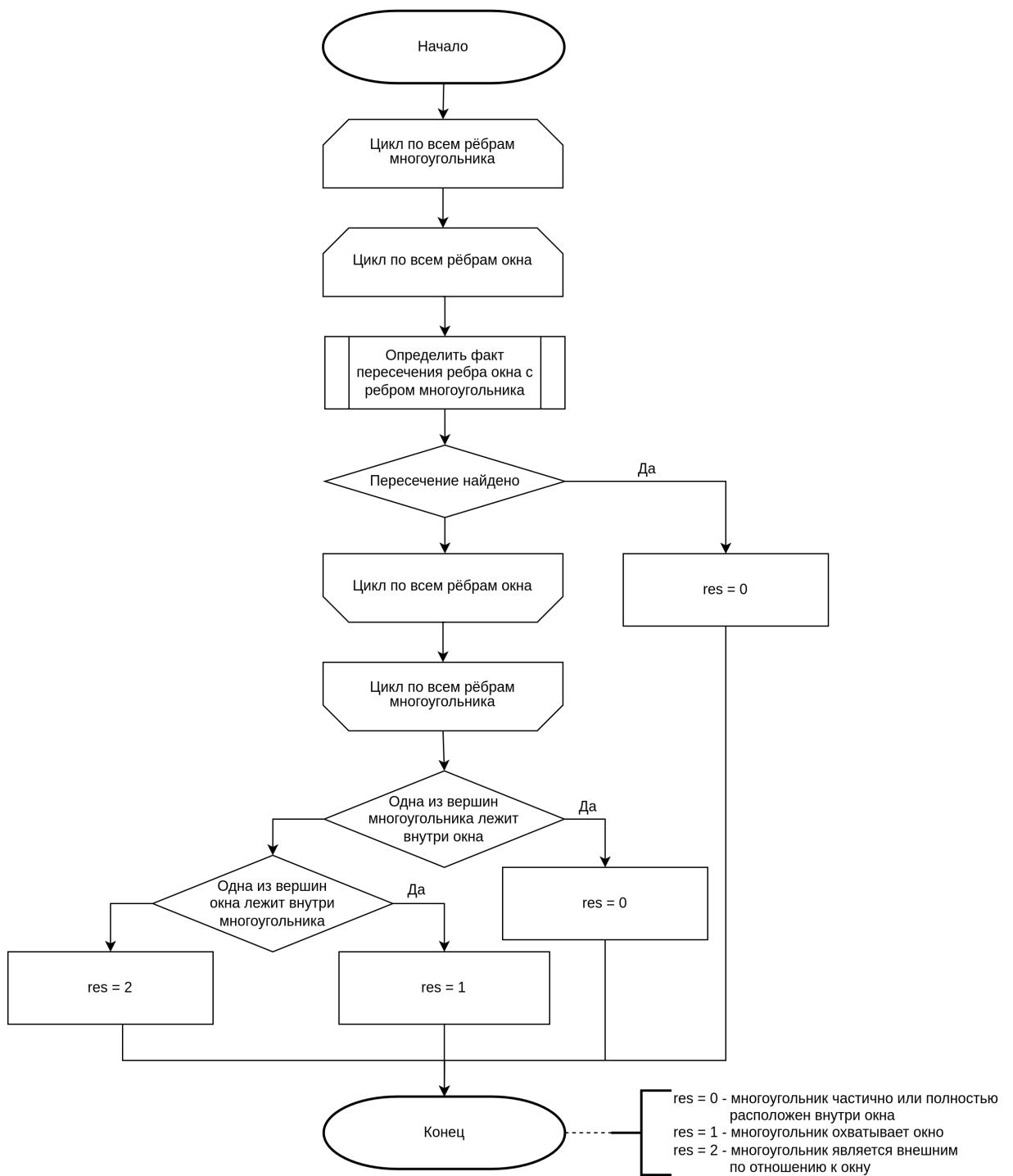


Рисунок Б.2 — Схема алгоритма определения взаимного расположения многоугольника относительно окна

Приложение В Пример исходных данных для работы программы

Листинг В.1 — Файл cube.obj

```
1 # Blender v3.2.2 OBJ File: 'cube.blend'
2 # www.blender.org
3 mtllib cube.mtl
4 o Cube
5 v 0.205056 -0.942683 -1.438507
6 v 0.896998 -1.434100 0.372495
7 v 0.861082 0.929069 -1.181257
8 v 1.553024 0.437651 0.629744
9 v -1.553024 -0.437651 -0.629744
10 v -0.861082 -0.929069 1.181257
11 v -0.896998 1.434100 -0.372495
12 v -0.205056 0.942683 1.438507
13 vt 0.625000 0.500000
14 vt 0.875000 0.500000
15 vt 0.875000 0.750000
16 vt 0.625000 0.750000
17 vt 0.375000 0.750000
18 vt 0.625000 1.000000
19 vt 0.375000 1.000000
20 vt 0.375000 0.000000
21 vt 0.625000 0.000000
22 vt 0.625000 0.250000
23 vt 0.375000 0.250000
24 vt 0.125000 0.500000
25 vt 0.375000 0.500000
26 vt 0.125000 0.750000
27 vn -0.3460 0.2457 -0.9055
28 vn 0.3280 0.9359 0.1286
29 vn -0.8790 0.2525 0.4044
30 vn 0.3460 -0.2457 0.9055
31 vn 0.8790 -0.2525 -0.4044
32 vn -0.3280 -0.9359 -0.1286
33 usemtl Material
34 s off
35 f 1/1/1 5/2/1 7/3/1 3/4/1
36 f 4/5/2 3/4/2 7/6/2 8/7/2
37 f 8/8/3 7/9/3 5/10/3 6/11/3
38 f 6/12/4 2/13/4 4/5/4 8/14/4
39 f 2/13/5 1/1/5 3/4/5 4/5/5
40 f 6/11/6 5/10/6 1/1/6 2/13/6
```

Листинг В.2 — Файл cube.mtl

```
1 # Blender MTL File: 'cube.blend'
2 # Material Count: 1
3
4 newmtl Material
5 Ns 359.999993
6 Ka 1.000000 1.000000 1.000000
7 Kd 0.038252 0.050353 0.800000
8 Ks 0.500000 0.500000 0.500000
9 Ke 0.000000 0.000000 0.000000
10 Ni 1.450000
```

```
11 | d 1.000000
12 | illum 2
```

Листинг B.3 — Файл cube.lgt

```
1 | 1000 1000 1000
2 | -1000 -1000 -1000
```

Приложение Г Изображения тестовых моделей

Разрешение дисплея: 320x240 пикселей.

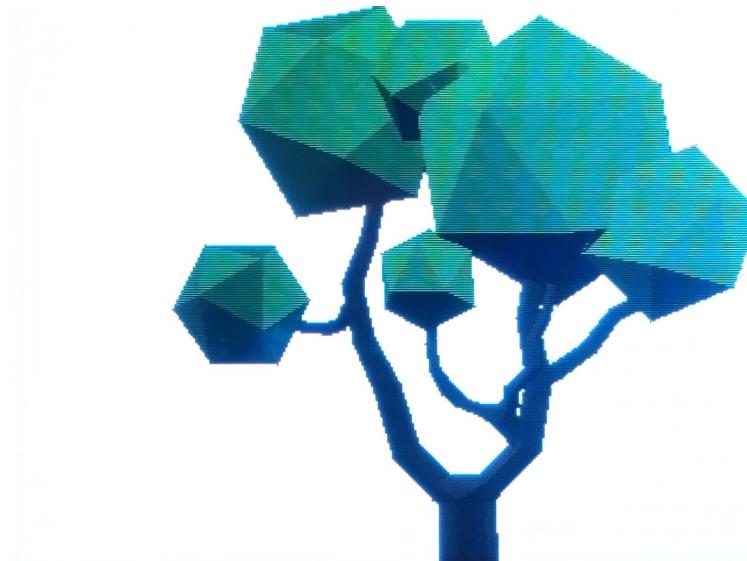


Рисунок Г.1 — Модель №1 (312 полигонов)



Рисунок Г.2 — Модель №2 (1172 полигона)



Рисунок Г.3 — Модель №3 (6228 полигонов)