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 КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Проектирование программно-аппаратного комплекса «Куб для визуализации трехмерного изображения»»

Студент ИУ7-53Б (Группа) _____ (Подпись, дата) Шпаковский П. А. (И. О. Фамилия)

Руководитель курсовой работы _____ (Подпись, дата) Строганов Ю. В. (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Компьютерная графика – это область информационных технологий, которая занимается созданием, обработкой, анализом и визуализацией графических объектов и изображений с использованием компьютерных методов и алгоритмов. Она имеет широкий спектр применений, включая развлечения, образование, научные исследования, медицинскую диагностику, архитектурное проектирование, визуализацию данных и многое другое.

Основные задачи, которые решает компьютерная графика.

1) Создание и редактирование изображений. Компьютерная графика позволяет создавать и редактировать графические объекты и изображения на компьютере. Это включает в себя рисование, моделирование форм и текстур, создание анимаций и спецэффектов.

2) Визуализация данных. В компьютерной графике данные могут быть представлены в визуальной форме, что делает их более понятными и удобными для анализа. Например, диаграммы, графики и трехмерная визуализация данных используются для отображения статистических и научных результатов.

3) 3D-моделирование и анимация. Компьютерная графика позволяет создавать трехмерные модели объектов и сцен, а также анимировать их движение и поведение. Это широко используется в фильмах, видеоиграх, архитектурном проектировании и медицинской визуализации.

Общим для всех этих задач является использование математических алгоритмов, программного обеспечения и аппаратных средств для создания, модификации и визуализации графических данных с целью достижения определенных визуальных и функциональных результатов.

Для визуализации трехмерных сцен на конкретной аппаратной платформе часто требуется разрабатывать индивидуальные программные решения, учитывающие особенности данного обеспечения.

Цель данной работы – спроектировать программно-аппаратный комплекс для построения моделей трехмерных объектов с использованием микроконтроллера.

Чтобы достичь поставленной цели, требуется решить следующие задачи:

— описать структуру трехмерной сцены, включая объекты, из которых состоит сцена, и определить формат задания исходных данных;

— выбрать наиболее подходящий из существующих алгоритмов трехмерной графики, позволяющих синтезировать изображение трехмерной сцены;

— реализовать выбранные алгоритмы построения трехмерной сцены;

— исследовать возможности микроконтроллеров в области визуализации трехмерной графики.

1 Аналитическая часть

В данной части проводится анализ объектов сцены и существующих алгоритмов построения изображений и выбор более подходящих алгоритмов для дальнейшего использования.

1.1 Описание объектов визуализируемой сцены

Трехмерная сцена состоит из нескольких компонентов.

1) Объекты сцены, представляющих из себя многогранники, расположенные в пространстве сцены. Каждый многогранник задается списком полигонов, где полигон описывается множеством точек в трехмерном пространстве.

2) Материалы и свойства объектов сцены. Каждый объект может иметь определенные свойства, включая цвет, текстуры, прозрачность, отражение и блеск. Эти свойства могут быть формализованы с помощью числовых параметров, которые определяют, как объект взаимодействует со светом и как он будет визуализирован.

3) Источники света, которые могут быть формализованы как направленные вектора.

4) Камера, через которую наблюдается данная сцена. Ее параметры могут включать положение, ориентацию, угол обзора и другие характеристики.

1.2 Анализ алгоритмов удаления невидимых линий и поверхностей

В процессе создания реалистичных изображений ключевой задачей является обеспечение видимости объектов и их частей, учитывая перекрытия между ними. Другими словами, необходимо определить, какие элементы сцены должны быть видны, а какие скрыты от наблюдателя. Эта задача решается с помощью двух групп алгоритмов [1].

1) Алгоритмы, оперирующие в объектном пространстве. Эти алгоритмы связаны с мировой или физической системой координат. Они предназначены для определения видимости объектов, учитывая их пространственное расположение. Однако такие методы требуют значительных

вычислительных ресурсов, которые зависят от сложности сцены и необходимой точности. Примерами таких алгоритмов могут быть алгоритм Робертса, алгоритмы на основе списков приоритетов и др.

2) Алгоритмы, оперирующие в пространстве изображения. Эти алгоритмы ориентированы на систему координат экрана или плоскости изображения, на которую проецируются объекты. Этот подход требует меньше вычислительных ресурсов по сравнению с первой группой, и его объем зависит от разрешения экрана и количества объектов на сцене. Примерами таких алгоритмов являются алгоритм Варнака, алгоритм Z-буфера и метод трассировки лучей.

1.2.1 Алгоритм Робертса

Алгоритм Робертса — это метод удаления невидимых линий и поверхностей в компьютерной графике, который действует в объектном пространстве [1]. Его основной целью является определение, какие линии (рёбра) объектов на сцене будут видны из точки наблюдения, а какие будут скрыты за другими объектами.

Алгоритм Робертса делится на 3 этапа.

1) На первом этапе каждое тело анализируется индивидуально с целью удаления нелицевых плоскостей. Для каждого тела и каждой грани тела вычисляется уравнение плоскости. После чего проверяется знак уравнения плоскости и формируется матрица тела.

2) Удаление из каждого тела тех ребер, которые экранируются всеми остальными телами в сцене. Если задано только одно тело, то алгоритм завершается.

3) На третьем этапе вычисляются отрезки, которые образуют новые ребра при протыкании телами друг друга.

За счет того, что алгоритм Робертса работает в объектном пространстве, получается достичь высокую точность результирующего изображения. Однако в данном алгоритме используется много трудоемких математических вычислений и сложность растет как квадрат числа объектов на сцене.

1.2.2 Алгоритм, использующий Z-буфер

Алгоритм, использующий Z-буфер, представляет собой один из наиболее простых и широко применяемых методов для удаления невидимых поверхностей. Этот алгоритм оперирует в пространстве изображения и основывается на идее использования двух буферов: буфера кадра и Z-буфера.

1) Буфер кадра (англ. Frame Buffer). Этот буфер представляет собой массив пикселей, в котором будут храниться атрибуты (например, цвет) каждого пикселя на экране.

2) Z-буфер (англ. Depth Buffer). Это отдельный буфер глубины, который будет хранить глубину (Z-координату) каждого пикселя на экране. Исходно он заполняется максимально возможными значениями глубины.

Алгоритм работает следующим образом.

1) Перед началом работы буфер кадра заполняется информацией о фоновом изображении, а Z-буфер заполняется минимальными значениями глубины.

2) Для каждого объекта на сцене:

   — полигоны объекта преобразуются в растровую форму;

   — для каждого пикселя текущего полигона проверяется его Z-координата (глубина) в сравнении с соответствующим значением в Z-буфере;

   — если Z-координата текущего пикселя меньше, чем значение в Z-буфере, то пиксель заносится в буфер кадра, а его Z-координата обновляется в Z-буфере.

3) На выходе получается изображение, в котором каждый пиксель находится в буфере кадра в соответствии с его видимостью и глубиной.

К преимуществам относятся простота реализации, которая позволяет достичь хороших результатов без сложной сортировки объектов. Также алгоритм эффективно обрабатывает сцены с большим количеством объектов.

К недостаткам можно отнести потребление большого объема памяти для хранения двух буферов, а также могут быть проблемы с прозрачностью и лестничным эффектом (ступенчатость на границах объектов).

1.2.3 Алгоритм Варнока

Алгоритм Варнока – это метод удаления невидимых поверхностей в компьютерной графике, который использует разбиение сцены на более мелкие части для определения видимости объектов. Алгоритм работает в пространстве изображения и анализирует область на экране дисплея (окно) на наличие в них видимых элементов [2].

Идея алгоритма Варнока заключается в разбиении экрана на рекурсивные прямоугольники, которые обозначают различные части сцены. Затем каждый прямоугольник проверяется на наличие видимых объектов. Если внутри прямоугольника объекты видимы, он разбивается на более мелкие прямоугольники, и процесс продолжается до достижения требуемого уровня детализации или пока не будут найдены видимые объекты.

Процесс алгоритма Варнока.

1) Разбиение экрана. Исходный экран разбивается на начальные прямоугольники. Эти прямоугольники могут представлять собой целый экран или другие крупные области.

2) Проверка видимости. Для каждого прямоугольника выполняется проверка видимости объектов, находящихся внутри него.

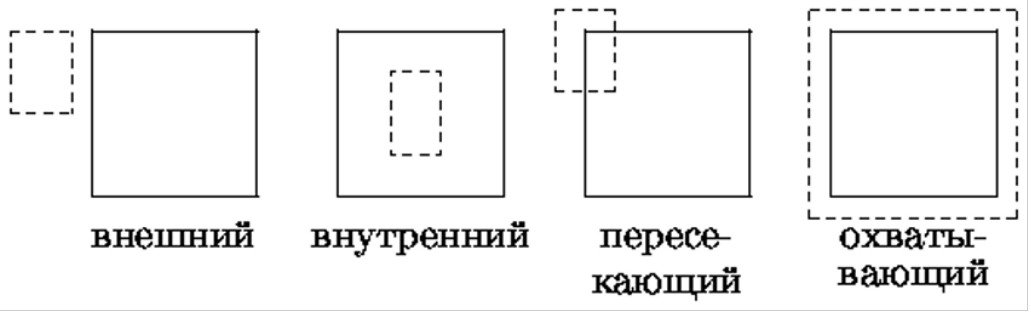

Рисунок 1 – Варианты взаимного расположения многоугольников [2]

3) Принятие решения. В зависимости от результата проверки видимости объектов внутри прямоугольника, он может быть разбит на более мелкие

подпрямоугольники или объединен с другими прямоугольниками. Этот процесс продолжается рекурсивно.

4) Заполнение буфера кадра. Для видимых прямоугольников или их частей выполняется заполнение буфера кадра атрибутами объектов.

5) Завершение. Процесс рекурсивного разбиения и проверки видимости продолжается до достижения требуемого уровня детализации или до тех пор, пока не будет достигнута наименьшая обрабатываемая единица.

Преимущества алгоритма Варнока включают возможность обработки сложных сцен, включая объекты с перекрывающимися поверхностями. Однако он может потребовать больших вычислительных ресурсов и времени, особенно при работе с детализированными сценами.

1.2.4 Алгоритм, использующий список приоритетов

Этот алгоритм базируется на упорядочивании объектов сцены в зависимости от их приоритета, который определяется их удаленностью от наблюдателя или глубиной в сцене. Идея заключается в начальном рендеринге объектов, находящихся на большем расстоянии от наблюдателя, и последующем перекрытии их ближайшими объектами.

Сначала происходит сортировка объектов по глубине, что создает первоначальный список приоритетов. Затем этот список подвергается коррекции через выполнение тестов на экранирование для каждой пары многоугольников в списке. Эти тесты могут быть достаточно затратными по времени и сложности реализации. Суть тестов заключается в определении, перекрывает ли один многоугольник другой, и, если да, то какой из них ближе к наблюдателю.

Важно отметить, что идентификация случаев пересечения и циклического перекрытия многоугольников в этом алгоритме является сложной задачей. Вследствие этого алгоритм может сталкиваться с ограничениями в эффективности и точности при обработке таких сцен, где объекты перекрывают друг друга в сложных сценариях.

1.2.5 Алгоритм трассировки лучей

Алгоритм трассировки лучей – это метод удаления невидимых поверхностей в компьютерной графике, который симулирует путь световых лучей,

падающих на объекты в сцене. Он создает фотореалистичные изображения, учитывая отражение, преломление и тени.

Предполагается, что наблюдатель находится на бесконечности положительной полуоси Z, и все световые лучи от наблюдателя параллельны этой оси. В ходе алгоритма обратной трассировки лучей, начиная от наблюдателя, генерируются лучи и находятся пересечения этих лучей с объектами сцены. Путем анализа этих пересечений определяется самая ближняя точка пересечения по оси Z, которая представляет видимую часть поверхности. Атрибуты этого объекта затем используются для определения характеристик пикселя, через который проходит этот световой луч.

Для создания реалистичных эффектов освещения алгоритм генерирует дополнительные лучи от точек пересечения к источникам света. Если на пути такого луча встречается непрозрачный объект, это означает, что данная точка находится в тени.

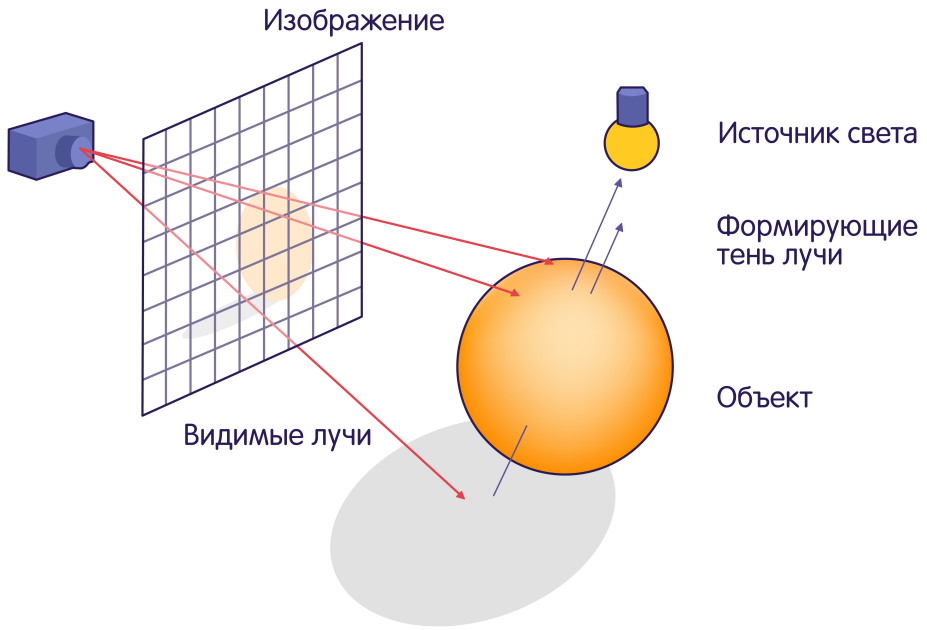

Рисунок 2 – Визуализация алгоритма трассировки лучей [3]

Основными преимуществами данного алгоритма являются реалистичность получаемого изображения и возможность анализировать наложение света. Однако из-за использования большого количества лучей производительность алгоритм снижается, поскольку для каждого луча необходимо производить поиск пересечений с объектами сцены.

1.3 Анализ алгоритмов закраски

1.3.1 Простая закраска

Применяется закрашивание, в котором каждая грань получает один уровень интенсивности, вычисляемый согласно закону Ламберта. В результате такой закраски все плоские поверхности, включая те, что аппроксимируют фигуры, подвергаются однородному окрашиванию. В случае фигур вращения это может вызвать появление ложных границ между поверхностями.

Этот метод обладает высокой производительностью, однако все пиксели на грани получают одинаковую интенсивность, что придает сцене нереалистичный вид. Несмотря на это, метод очень прост в реализации и не требует значительных вычислительных ресурсов.

1.3.2 Закраска по Гуро

Закраска по методу Гуро основана на билинейной интерполяции интенсивностей на поверхности грани. Этот метод создает иллюзию плавной криволинейной поверхности, так как интенсивность вычисляется для каждой точки грани, что способствует мягкому переходу между цветами.

Закраска по методу Гуро хорошо сочетается с простой моделью освещения, основанной на диффузном отражении света. Она значительно улучшает качество изображения, придавая ему более реалистичный вид.

Однако у этого метода есть свои недостатки:

— возникает эффект полос Маха – артефакт, проявляющийся в виде полос вдоль граней изображения;

— из-за сглаживания интенсивностей может быть утрачена четкая граница между гранями многогранника, что в некоторых ситуациях может привести к неверному визуальному результату.

Процедура закрашивания граней методом Гуро состоит из четырех последовательных этапов.

1) Вычисление граничных нормалей. На этом этапе определяются нормали для каждой отдельной грани. Нормаль – это вектор, перпендикулярный к поверхности грани и указывающий направление в пространстве.

2) Определение вершинных нормалей. Затем определяются нормали в вершинах объекта. Нормаль в вершине вычисляется как среднее значение нормалей граней, смежных с данной вершиной. Это позволяет создать плавные переходы между нормалями граней и учесть геометрическую структуру объекта.

3) Вычисление яркостей в вершинах. Исходя из нормалей в вершинах, определяются значения яркостей (интенсивностей) в этих вершинах, согласно выбранной модели отражения света. Это позволяет учесть, как свет падает на разные части объекта.

4) Закрашивание граней. Наконец, происходит закрашивание полигонов граней. Каждая грань получает цвет, который вычисляется путем линейной интерполяции значений интенсивностей в вершинах этой грани. Это создает плавные переходы цветов на поверхности и добавляет визуальную реалистичность объекту.

1.3.3 Закраска по Фонгу

Закраска по модели Фонга, по своей концепции, схожа с закраской по методу Гуро, но имеет отличие в том, как интерполируются значения. В методе Гуро производится интерполяция значений интенсивности для всех точек полигона, тогда как в методе Фонга интерполируются векторы нормалей граней, и на основе этих нормалей определяется интенсивность для каждой точки [4].

Этапы алгоритма закраски по Фонгу следующие.

1) Вычисление нормалей для каждой грани.

2) На основе нормалей граней определяются нормали в вершинах. В каждой точке грани интерполируется вектор нормали, учитывая соседние вершины.

3) С использованием направления векторов нормали определяется цвет точек грани в соответствии с выбранной моделью освещения.

Алгоритм закраски по модели Фонга требует больше вычислительных ресурсов по сравнению с предыдущими методами, но он позволяет достичь

лучшей локальной аппроксимации кривизны поверхности. Это приводит к более реалистичному изображению, так как метод учитывает изменения в направлениях нормалей на поверхности, что способствует созданию более точных эффектов освещения.

1.4 Анализ существующих микроконтроллеров

При проектировании программно-аппаратного комплекса немаловажную роль играет выбор микроконтроллера и соответствующей периферии. Существует множество популярных семейств микроконтроллеров, каждое из которых имеет свои особенности и применения. Выбор конкретной модели зависит от требований проекта, бюджета, опыта разработчика и других факторов. Каждое из семейств микроконтроллеров имеет свои ограничения, которые следует учитывать при разработке системы.

1.4.1 STM32

Семейство микроконтроллеров STM32 [5] производится компанией STMicroelectronics и является одним из самых популярных среди разработчиков встраиваемых систем. STM32 представляет собой широкий спектр микроконтроллеров, предназначенных для различных приложений и задач.

STM32 микроконтроллеры основаны на ядре ARM Cortex-M, что обеспечивает высокую производительность и эффективное энергопотребление. Существуют разные версии STM32 с разными ядрами, такие как Cortex-M0 [6], Cortex-M3 [7], Cortex-M4 [8] и Cortex-M7 [9], каждое из которых имеет разную производительность и возможности.

Микроконтроллеры обычно имеют богатый набор встроенных периферийных устройств, таких как UART, SPI, I2C, GPIO, таймеры, АЦП и многое другое. Данные микроконтроллеры находят широкое применение в промышленных системах, автомобильной электронике, медицинском оборудовании, умных домах, робототехнике и многих других областях.

1.4.2 ESP32

Семейство микроконтроллеров ESP32 [10] разработано компанией Espressif Systems и представляет собой популярную линейку микроконтроллеров, спроектированных специально для Интернета вещей (англ. Internet of

things). Они обладают выдающимися характеристиками, обеспечивая высокую производительность и богатый набор функций.

ESP32 использует двухъядерное процессорное ядро на базе архитектуры Xtensa LX6 [11] от компании Tensilica. Это обеспечивает высокую производительность и возможность многозадачности. Тактовая частота процессора обычно находится в диапазоне 80-240 МГц, в зависимости от конкретной модели. Имеется встроенная флеш-память (от 4 МБ до 16 МБ) и оперативная память (от 512 КБ до 8 МБ).

ESP32 широко используется в различных проектах IoT, включая умные дома, сенсорные устройства, системы мониторинга, автоматизацию и многие другие приложения.

1.4.3 Raspberry Pi

Семейство микроконтроллеров Raspberry Pi [12] представляет собой серию одноплатных компьютеров и микроконтроллеров, разрабатываемых фондом Raspberry Pi Foundation. Эти устройства получили широкую популярность во всем мире и широко используются в образовании, промышленности. Raspberry Pi имеет активное сообщество разработчиков, что обеспечивает доступ к множеству бесплатных ресурсов, библиотекам и поддержке.

Raspberry Pi использует разные архитектуры в зависимости от модели. Ранние модели (например, Raspberry Pi 1) основаны на архитектуре ARMv6 [13] или ARMv7 [14], а более новые (например, Raspberry Pi 3 и 4) используют архитектуру ARM Cortex-A [15].

Raspberry Pi находит применение в различных областях, включая обучение информатике и электронике, создание умных домов и умных городов, автоматизацию, робототехнику, медиацентры и многое другое. Также данное семейство известено своей доступной ценой, что делает его отличным выбором для проектов с ограниченным бюджетом.

Raspberry Pi Pico – это плата с микроконтроллером, разработанная фондом Raspberry Pi Foundation. Она представляет собой компактное устройство, которое можно использовать для разработки встраиваемых систем и микроконтроллерных проектов. В составе данной платы используется микроконтроллер RP2040 [16]. RP2040 основан на двухъядерном процессоре ARM Cortex-M0+ и обеспечивает высокую производительность и эффективное энергопотребление.

Микроконтроллер способен работать на частотах до 133 МГц. У Raspberry Pi Pico есть 264 Кб SRAM и 2 Мб флеш-памяти. Также он оборудован 26 портами GPIO, которые можно настроить для ввода или вывода. Эти порты поддерживают различные интерфейсы, включая UART, SPI, I2C и PWM.

1.4.4 NXP/Freescale

Семейство микроконтроллеров NXP [17] (ранее Freescale) включает в себя разнообразные микроконтроллеры, разработанные компанией NXP Semiconductors (ранее Freescale Semiconductor) для широкого спектра встраиваемых систем и приложений. Многие микроконтроллеры NXP/Freescale основаны на ядрах ARM Cortex-M, таких как Cortex-M0, Cortex-M3, Cortex-M4 и Cortex-M7.

Частота работы микроконтроллеров может варьироваться от нескольких десятков мегагерц до нескольких сотен мегагерц в зависимости от модели. Данное семейство обладает большим спектром встроенных периферийных устройств, что дает ему преимущество перед другими микроконтроллерами.

Микроконтроллеры NXP/Freescale находят применение в широком спектре областей, включая автомобильную электронику, промышленные системы, медицинское оборудование, умные дома и многие другие. Компания NXP и ее сообщество разработчиков предоставляют обширную документацию, библиотеки и техническую поддержку для своих микроконтроллеров.

1.5 Критерии выбора алгоритмов трехмерной визуализации

При выборе алгоритмов компьютерной графики для визуализации трехмерной сцены следует учитывать особенности платформы, с помощью которой будет решаться задача. В данном случае необходимо рассмотреть возможности микроконтроллеров.

1) Объем оперативной памяти на микроконтроллере достаточно ограничен. Обычно он составляет не больше нескольких мегабайт. Поэтому при выборе алгоритмов стоит учитывать размер используемых структур данных и затраченной памяти в целом.

2) Вычислительная мощность микроконтроллеров также невысокая, поэто-

му при выборе алгоритма стоит учитывать его простоту и эффективность работы. Немаловажной будет являться сложность работы алгоритма, по возможности, она должна быть линейной в зависимости от количества объектов сцены.

3) Объем встроенной постоянной памяти в микроконтроллерах небольшой и составляет не более 2 мегабайт. Исходя из этого размер исполняемого файла должен быть меньше, поэтому стоит выбирать алгоритмы, реализация которых более лаконичная и компактная.

Для поставленной задачи критерий, связанный с объемом оперативной памяти, будет приоритетным.

1.6 Вывод из аналитической части

Проанализировав алгоритмы удаления невидимых линий и поверхностей и различные алгоритмы закраски, а также рассмотрев критерии выбора наиболее оптимального алгоритма для поставленной задачи визуализации трехмерной сцены с использованием микроконтроллера, можно сделать вывод, что алгоритм Варнока и алгоритм простой закраски подходят лучше всего. Алгоритм Варнока использует оптимальное количество памяти и требует меньшей вычислительной мощности, по сравнению с другими алгоритмами. Простая закраска наименее ресурсоемкая из всех алгоритмов закраски, при этом позволяет достичь поставленной цели. При выборе алгоритмов ключевую роль играли особенности аппаратной платформы.

2 Конструкторская часть

В данном разделе будут описаны алгоритмы и структуры данных, выбранные для решения поставленной задачи, будет разработана структура программно-аппаратного комплекса.

2.1 Структуры данных

Чтобы формализовать алгоритм синтеза изображения в программе, необходимо определить структуры данных, которые будут в ней использоваться. В данной работе приняты следующие соглашения. Трёхмерные модели являются полигональными, тогда сцену можно представить в виде массива многоугольников (полигонов). Многоугольник включает в себя следующие данные:

— количество вершин;

— массив x и y координат вершин;

— коэффициенты уравнения поверхности, несущей данный многоугольник заданного в виде $a \cdot x + b \cdot y + c \cdot z = d$;

— цвет в цветовой модели RGB [18].

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 верхней и правой нижней вершин окна.

Структура камеры содержит:

— позицию камеры в объектном пространстве;

— координаты точки, в которую направлен обзор камеры;

— вектор, направленный вверх;

— вектор, направленный вправо.

Сцена состоит из объектов в пространстве, камеры и источников света.

2.2 Трехмерные преобразования

Для корректной работы алгоритмов удаления невидимых линий и поверхностей сначала необходимо провести преобразования объектов сцены, заданных в объектном пространстве.

2.2.1 Пространство камеры

Первым преобразованием является переход из системы координат глобального пространоства в пространство камеры (англ. Camera view) [19].

$$M_{view} = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (1)$$

где U – вектор, задающий камеру и направленный вверх, R – вектор, задающий камеру и направленный вправо, D – вектор направления обзора камеры, P – вектор позиции камеры. M_{view} – итоговая матрица преобразования произвольного вектора из глобальной системы координат в систему координат камеры.

2.2.2 Перспективные преобразования

Для построения реалистичных изображений необходимо учитывать перспективу трехмерной сцены. Для этого объекты, находящиеся в пространстве камеры, подвергаются перспективному преобразованию.

$$M_{proj} = \begin{bmatrix} \frac{1}{ar \cdot \mathrm{tg}\,\frac{\alpha}{2}} & 0 & 0 & 0 \\ 0 & \frac{1}{\mathrm{tg}\,\frac{\alpha}{2}} & 0 & 0 \\ 0 & 0 & \frac{-NearZ-FarZ}{NearZ-FarZ} & \frac{2 \cdot FarZ \cdot NearZ}{NearZ-FarZ} \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad (2)$$

где ar – соотношение сторон изображения (англ. aspect ration), $NearZ$ и $FarZ$ – координаты Z, ограничивающие область преобразования, α – угол обзора.

M_{proj} – матрица перспективного преобразования [20].

Итоговое преобразование вектора из глобальной системы координат, в систему координат, требующуюся для синтеза изображения можно записать в виде формулы 3:

$$V_{result} = M_{proj} \cdot M_{view} \cdot V_{global}, \tag{3}$$

где V_{result} – итоговый вектор, V_{global} – вектор в глобальной системе координат [21].

2.3 Структура аппаратного комплекса

Макет устройства представляет собой макетную плату, к которой подключены микроконтроллер Raspberry Pi Pico, шесть TFT дисплеев с разрешением 240x240 пикселов, а также кнопка перезапуска микроконтроллера. Периферия подключена к шине SPI с тактовой частотой 40 МГц. Для обеспечения выбора одного из шести дисплеев для отрисовки текущего кадра используется отдельный пин Chip Select [22]. Питание платы осуществляется за счет USB провода, подключенного к микроконтроллеру.

2.4 Алгоритм работы программно-аппаратного комплекса

Алгоритм работы программно-аппаратного комплекса представлен в виде диаграммы, оформленной в соответствии с нотацией IDEF0 и отражающей общую декомпозицию алгоритма [23].

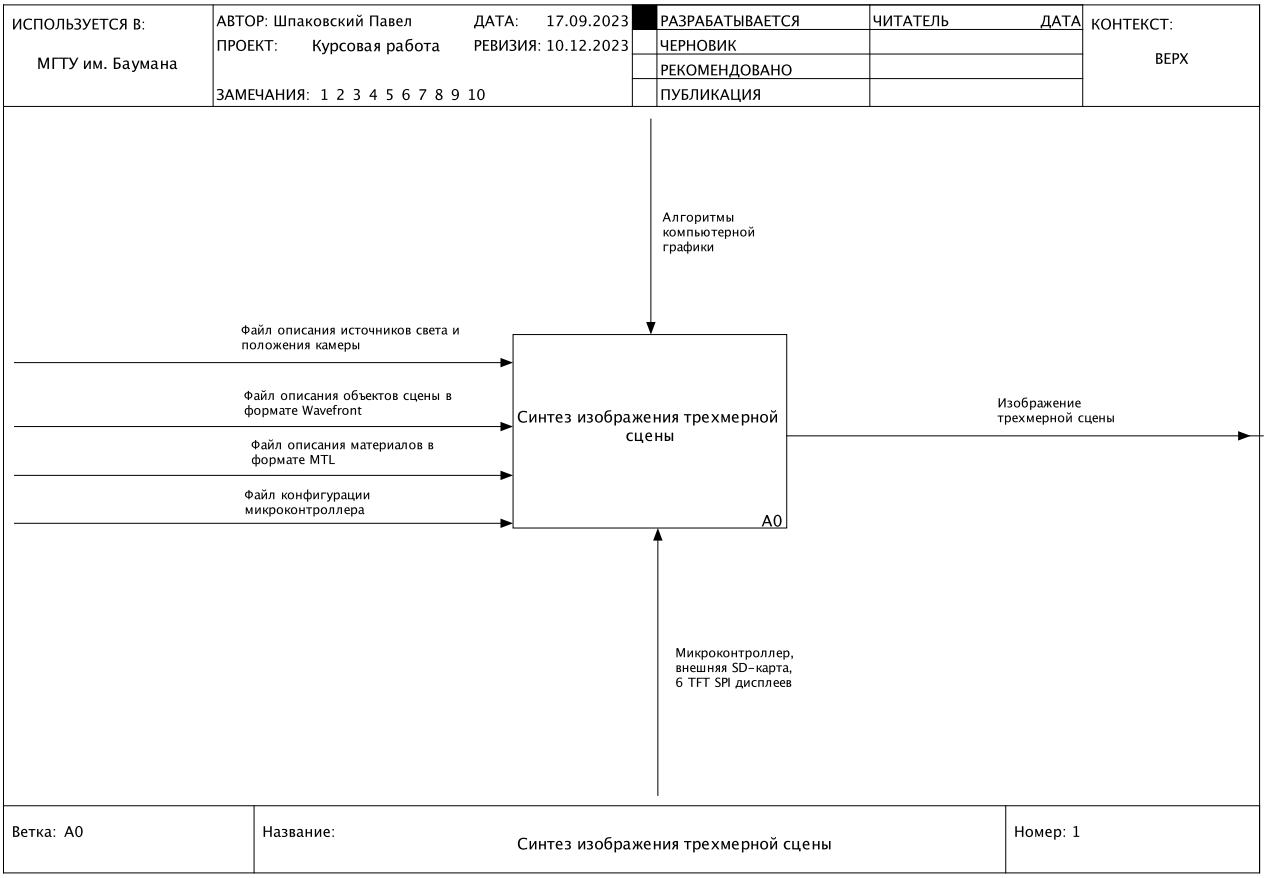

Рисунок 3 – Функциональная схема программно-аппаратного комплекса, декомпозиция верхнего уровня

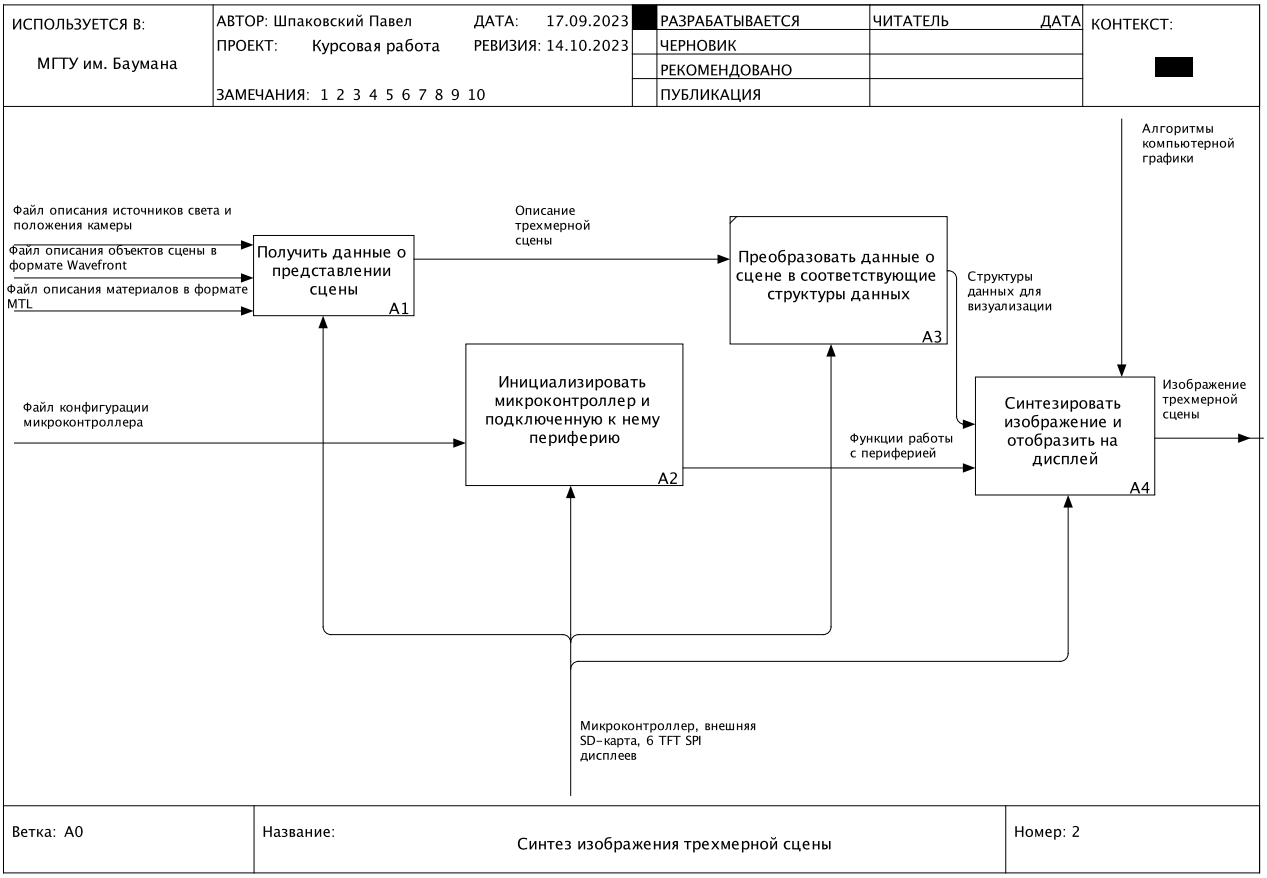

Рисунок 4 – Функциональная схема программно-аппаратного комплекса, декомпозиция уровня A0

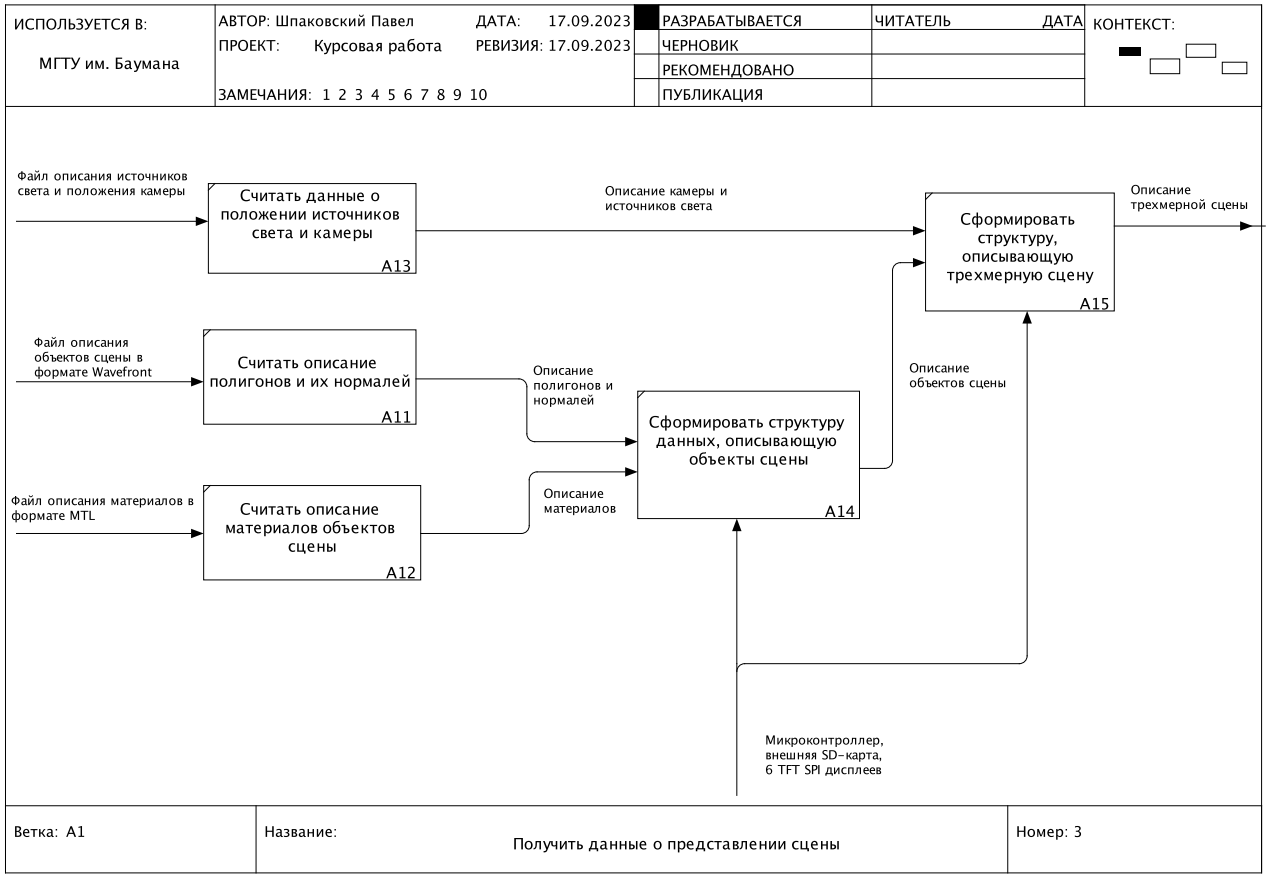

Рисунок 5 – Функциональная схема программно-аппаратного комплекса, декомпозиция уровня A1

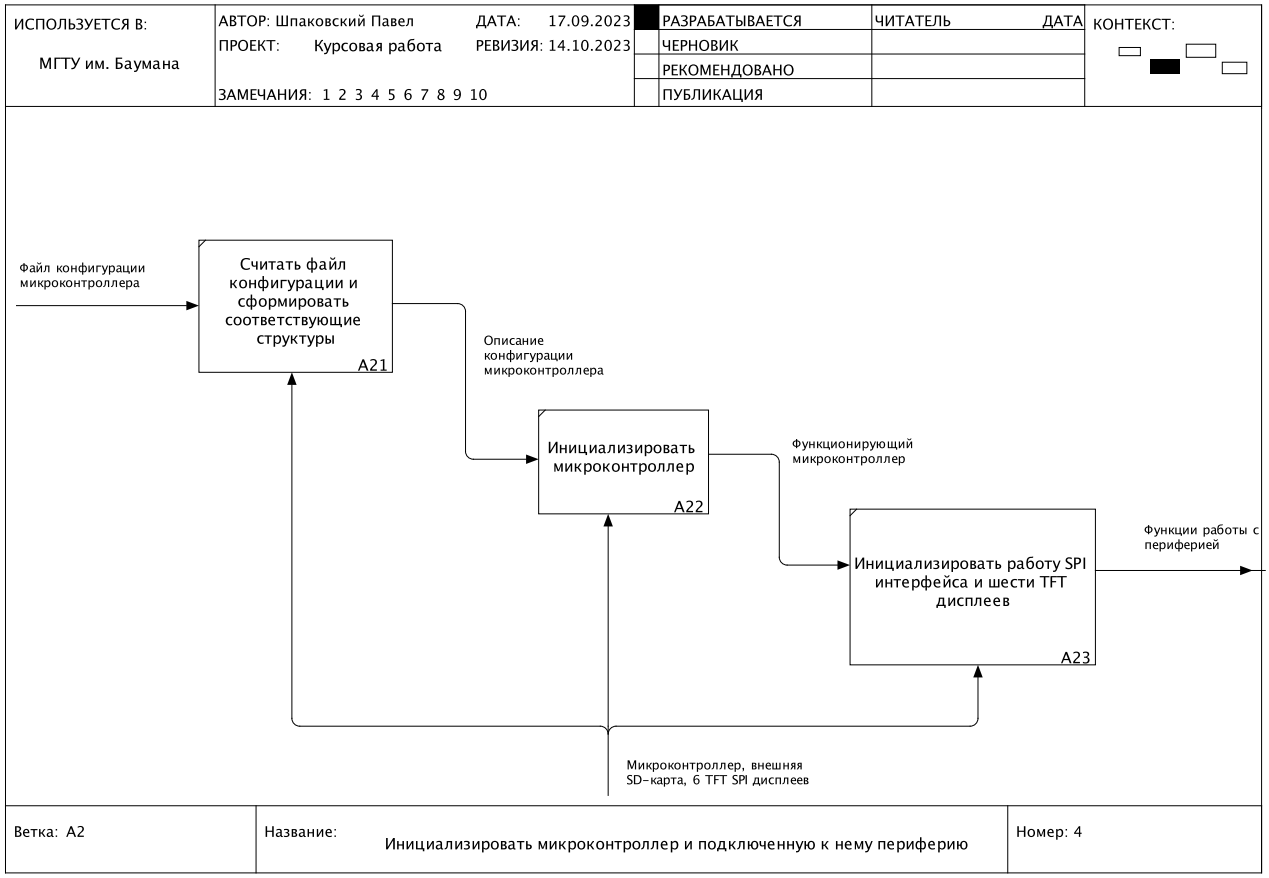

Рисунок 6 – Функциональная схема программно-аппаратного комплекса, декомпозиция уровня A2

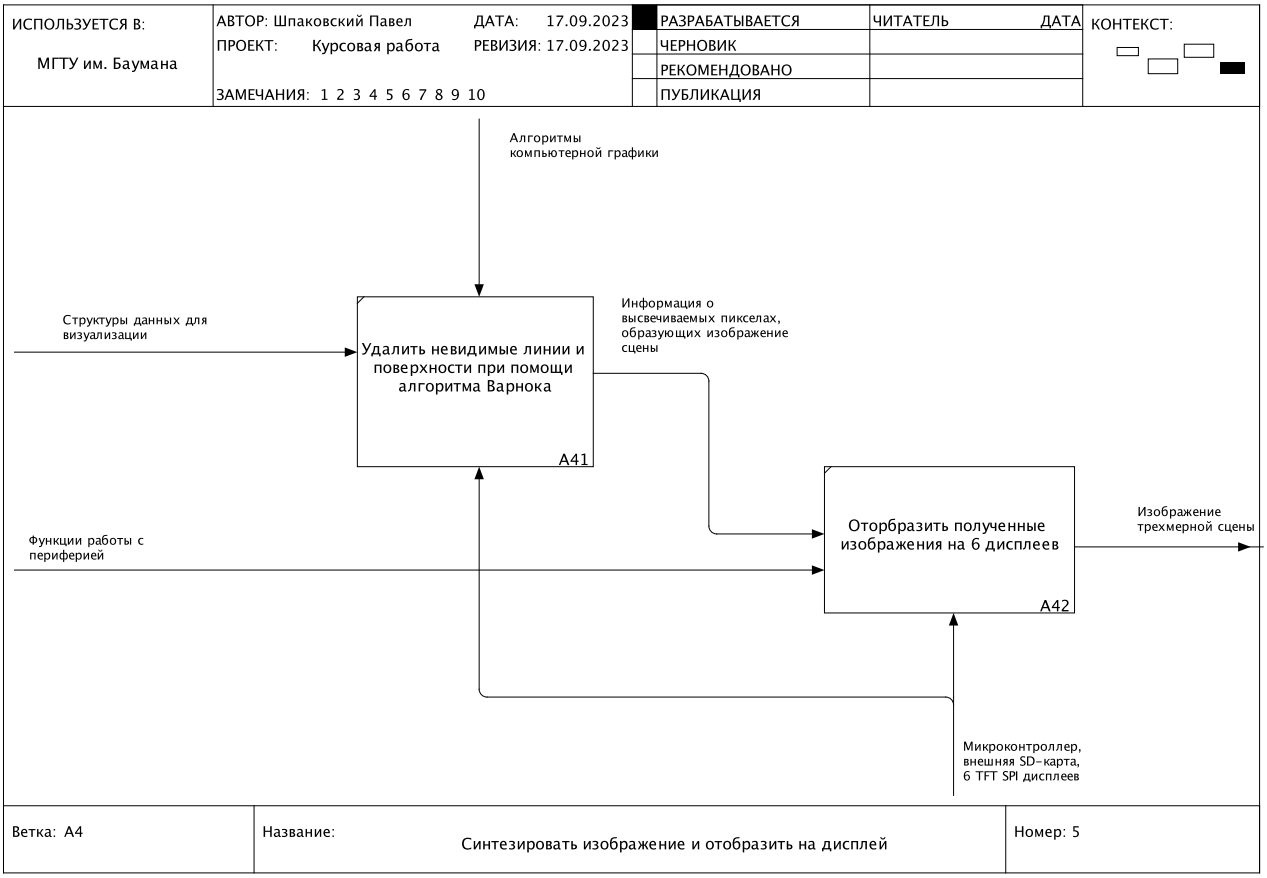

Рисунок 7 – Функциональная схема программно-аппаратного комплекса, декомпозиция уровня A4

2.5 Алгоритм Варнока

Для удаления невидимых линий и поверхностей был выбран алгоритм Варнока, который основан на рекурсивном разбиении окон. Окно представляет из себя некоторую прямоугольную подобласть итогового изображения. В данной работе будут рассматриваться две итерационные реализация алгоритма. Классическая реализация алгоритма Варнока – реализация, при которой начальное окно разделяется на четыре равные части на каждом уровне рекурсии. Такая реализация ползволяет решить поставленную в данной работе задачу, однако такой метод решения не является эффективным по времени исполнения. Будет также рассмотрен модифицированный алгоритм Варнока, использующий возможности аппаратной платформы более эффективно.

2.5.1 Классический алгоритм Варнока

Данный алгоритм представляет из себя итеративную реализацию, использующую стек для избежания рекурсии. Изображение для каждого из

шести экранов на макете устройства синтезируется и отображается на экран последовательно, что приводит к большим задержкам отрисовки.

Схема алгоритма Варнока представлена на рисунке 8.

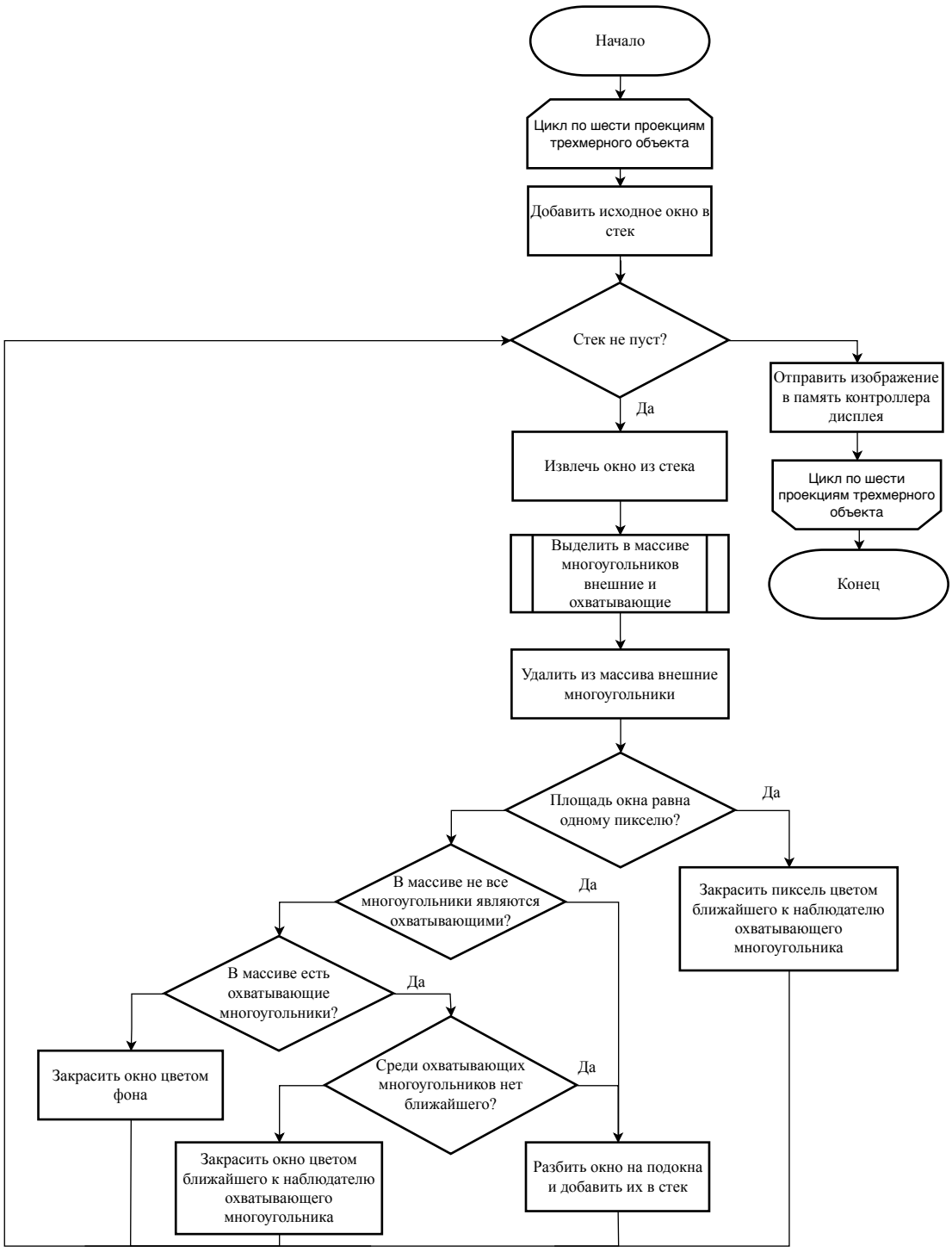

Рисунок 8 – Схема работы алгоритма Варнока

2.5.2 Модифицированный алгоритм Варнока

Модифицированная реализация алгоритма позволяет использоваться такую технологию, как прямой доступ к памяти (англ. Direct memory access) [24]. Поскольку в микроконтроллере малый объем оперативной памяти, то буфер кадра будет разделен на шесть сегментов, каждый из которых отвечает за прямоугольную подобласть экрана. Таким образом изображение будет синтезироваться для каждого экрана по частям, при этом каждая область изображения отправляется на контроллер экрана с использованием прямого доступа к памяти. Преимуществом такого подхода является то, что прямой доступ к памяти не задействует процессорное время, следовательно при отправке части изображения, микроконтроллер может продолжать синтезировать оставшиеся области.

Схема модифицированного алгоритма Варнока представлена на рисунке 9.

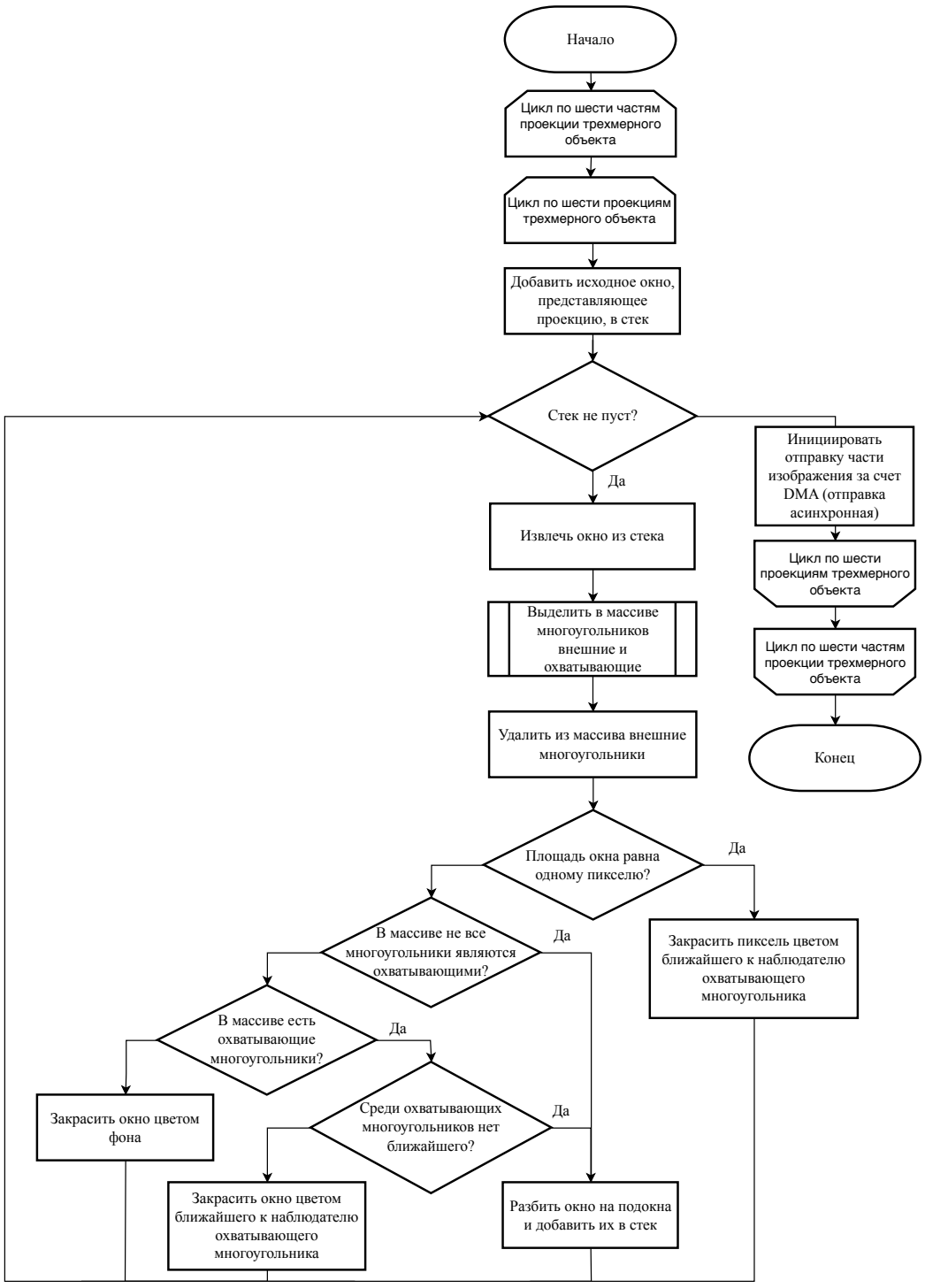

Рисунок 9 – Схема работы модифицированного алгоритма Варнока

2.6 Вывод из конструкторской части

В данном разделе были описаны алгоритмы и структуры данных, выбранные для решения поставленной задачи и отвечающие заданным требованиям.

3 Технологическая часть

В данном разделе будут рассмотрены детали реализации программно-аппаратного комплекса, описанного в конструкторской части работы, и приведены примеры работы макета устройства.

3.1 Средства реализации

Для реализации данной курсовой работы выбран язык программирования C++ [25]. Язык предоставляет высокую производительность при использовании в компьютерной графике. Также используются вставки на языке C для работы с низкоуровневым интерфейсом микроконтроллера и обеспечения коммуникации с периферией. Драйвер дисплея с чипом ST7789 в своей реализации использует язык C.

Для отладки программы используется интерфейс SWD (Serial Wire Debug) [26] и программа OpenOCD [27], предоставляющая возможность удаленной отладки с персонального компьютера.

Взаимодействие пользователя с программой осуществляется с помощью консоли, которая реализуется за счет протокола UART. Для соединения компьютера и микроконтроллера используется преобразователь UART в USB.

3.2 Особенности реализации

Макет устройства использует два протокола взаимодействия с периферией: SPI и UART. UART применяется для реализации консольного пользовательского интерфейса для взаимодействия с трехмерной сценой. SPI для работы дисплеев.

При конструировании устройства была выявлена проблема, связанная с питанием. Для работы шести дисплеев требуется большая мощность источника питания, по сравнению с той, которая предоставляется микроконтроллером (3.3 В). Для работы макета устройства необходимо использовать стационарный источник питания с напряжением 12 В. В связи с отсутствием такого источника, аппаратный комплекс был переработан и на данный момент количество дисплеев равно двум. Однако в исходный код программы заложена возможность визуализации трехмерной сцены на шести дисплеях. Таким образом, при наличии стационарного источника питания, программно-аппаратный

комплекс будет работать корректно.

3.3 Форматы описания трехмерной сцены

Геометрическое описание объектов сцены представляется в формате Wavefront OBJ [28]. В данной работе используется полигональная модель представления трехмерных объектов. Формат obj хранит текстовую информацию о вершинах полигонов, их нормалях и о материале полигона.

Материал полигона задается в формате Wavefront MTL [29]. В случае использования простой закраски при визуализации сцены необходимо получить из файла только компоненту, задающую цвет полигона.

Сцена представляет из себя объекты, расположенные в пространстве, камеру и источники освещения. Эти данные представлены в виде текстового файла формата scene. Данный формат схож с форматами obj и mtl.

В листинге 1 приведен пример файла описания трехмерной сцены.

Листинг 1 – Пример файла описания трехмерной сцены

```
# path to .obj file
o models/sphere.obj

# path to .mtl file
m models/sphere.mtl

# light positions
l -1.0 -1.0 -1.0
l -1.0 1.0 1.0
l 1.0 1.0 1.0

# camera position vector
cp -10.0 10.0 -15.0
# camera target vector (look at)
ct 0.0 0.0 0.0
# camera up vector
cu 0.0 1.0 0.0
```

3.4 Модули программы

Разработанный программный комплекс разбит на следующие модули:

— main.cpp – файл, содержащий точку входа в программу. В нем происхо-

дит обработка команд от пользователя и основной цикл визуализации сцены;

— loader.cpp – файл, содержащий функции для загрузки информации об объектах сцены из файлов в форматах obj, mtl, scene;

— dataset.cpp – сгенерированный файл, включающий в себя описания нескольких объектов сцены, которые позже будут скопировны во флеш память микроконтроллера вместе с машинным кодом;

— math – модуль, реализующий математические операции и структуры данных;

— render – модуль, отвечающий за отрисовку трехмерной сцены;

— scene – модуль, описывающий структуры данных для описания сцены;

— lib – сторонние библиотеки, использующиеся в работе;

— desktop – реализация алгоритмов визуализации для персонального компьютера.

3.5 Реализация алгоритмов визуализации

В листинге 2 приведен код реализации алгоритма Варнока удаления невидимых граней.

Листинг 2 – Листинг кода реализации алгоритма Варнока

```
void warnock_render(display_t *display, const window
   &full_window,
    const uint16_t bg_color,
  void set_pixel(display_t *, point2, uint16_t)) {
  std::stack<window> stack;
  stack.push(full_window);
  while (!stack.empty()) {
    window current_window = stack.top();
    stack.pop();
    size_t index = 0;
    size_t disjoint_cursor = 0;
    size_t surrounding_cursor = current_window.polygons.size;
    while (index < surrounding_cursor) {
      polygon polygon = current_window.polygons.data[index];
      relationship rel = check_relationship(polygon,
         current_window);
      if (rel == relationship::disjoint) {
        std::swap(current_window.polygons.data[index++],
             current_window.polygons.data[disjoint_cursor++]);
      } else if (rel == relationship::surrounding) {
        std::swap(current_window.polygons.data[index],
             current_window.polygons.data[--surrounding_cursor]);
      } else {
        ++index;
      }
    }
    array<polygon> visible = {current_window.polygons.data +
       disjoint_cursor,
                 current_window.polygons.size - disjoint_cursor};
    uint16_t window_width = current_window.end.x -
       current_window.begin.x;
    uint16_t window_height = current_window.end.y -
       current_window.begin.y;
```

Листинг 3 – Продолжение листинга кода реализации алгоритма Варнока

```
    if (window_width == 1 && window_height == 1) {
      if (visible.size == 0) {
        set_pixel(display, current_window.begin, bg_color);
      } else {
        fill_pixel(display, current_window.begin, visible,
          set_pixel);
      }
    } else if (surrounding_cursor != disjoint_cursor) {
      split_window(stack, current_window, visible);
    } else {
      if (visible.size == 0) {
        fill_window(display, current_window, bg_color,
          set_pixel);
        continue;
      }
      std::pair<bool, polygon> result =
        find_cover_polygon(current_window, visible);
      if (result.first) {
        fill_window(display, current_window,
          result.second.color, set_pixel);
      } else {
        split_window(stack, current_window, visible);
      }
    }
  }
}
```

Переданное окно делится на 4 равных части и они заносятся в стек. Данный процесс длится до тех пор, пока объект, попадающий в окно не станет тривиально видимым или тривиально невидимым. Итеративная реализация алгоритм позволяет уменьшить объем используемой памяти.

3.5.1 Классический алгоритм Варнока

В листинге 4 приведен код, использующий реализацию классического алгоритма Варнока, без использования аппаратных средств ускорения визуализации. Предварительное разбиение окна на шесть частей не производится.

Листинг 4 – Классический алгоритм Варнока

```
for (size_t i = 0; i < 6; i++) {
    warnock_render(&displays[i],
       {{-displays[i].width / 2, -displays[i].height / 2},
        {displays[i].width / 2, displays[i].height / 2},
        state.polygons},
       BLACK, set_pixel);
    GFX_flush(&displays[i]);
}
```

3.5.2 Модифицированный алгоритм Варнока

В листинге 5 приведен код, использующий реализацию модифицированного алгоритма Варнока, который использует аппаратное ускорение с помощью прямого доступа к памяти. Предварительно окно разбивается на шесть частей для более эффективной пересылки данных по DMA.

Листинг 5 – Модифицированный алгоритм Варнока

```
for (size_t i = 0; i < 6; i++) {
    for (size_t j = 0; j < 6; j++) {
        window = windows[j][i];
        warnock_render(&displays[j], window, BLACK, set_pixel);
        GFX_flush_block(&displays[j], window.begin.x +
           displays[j].width / 2,
                        window.begin.y + displays[j].height / 2,
                        displays[j].width / 3,
                           displays[j].height / 2);
    }
}
```

3.6 Пользовательский интерфейс

В данной работе был реализован консольный пользовательский интерфейс. Для управления визуализируемой сценой используется набор текстовых команд, каждая из которых принимает ряд аргументов. Пользователь может загрузить новую трехмерную сцену по ее названию или вывести список всех доступных сцен в программе. Также возможно вращение камеры на заданный угол, приближение камеры и ее перемещение.

Поддерживаются следующие команды:

— help – вывод информации о командах;

— models – вывод списка названий доступных трехмерных сцен;

— load <название сцены> – загрузка сцены по ее названию;

— camera set cp <x, y, z> ct <x, y, z> cu <x, y, z> – установка камеры по трем векторам направлений;

— camera rotate <rx, ry, rz> – вращение камеры, где rx, ry, rz - углы поворота по осям в градусах;

— camera scale <k> – масштабирование камеры, где k - коэффициент масштабирования;

— camera reset – сброс настроек камеры к значению по умолчанию.

3.7 Пример работы программно-аппаратного комплекса

На рисунке 10 представлен пример визуализации трехмерной сцены, включающей две низкополигональные сферы и три источника света.

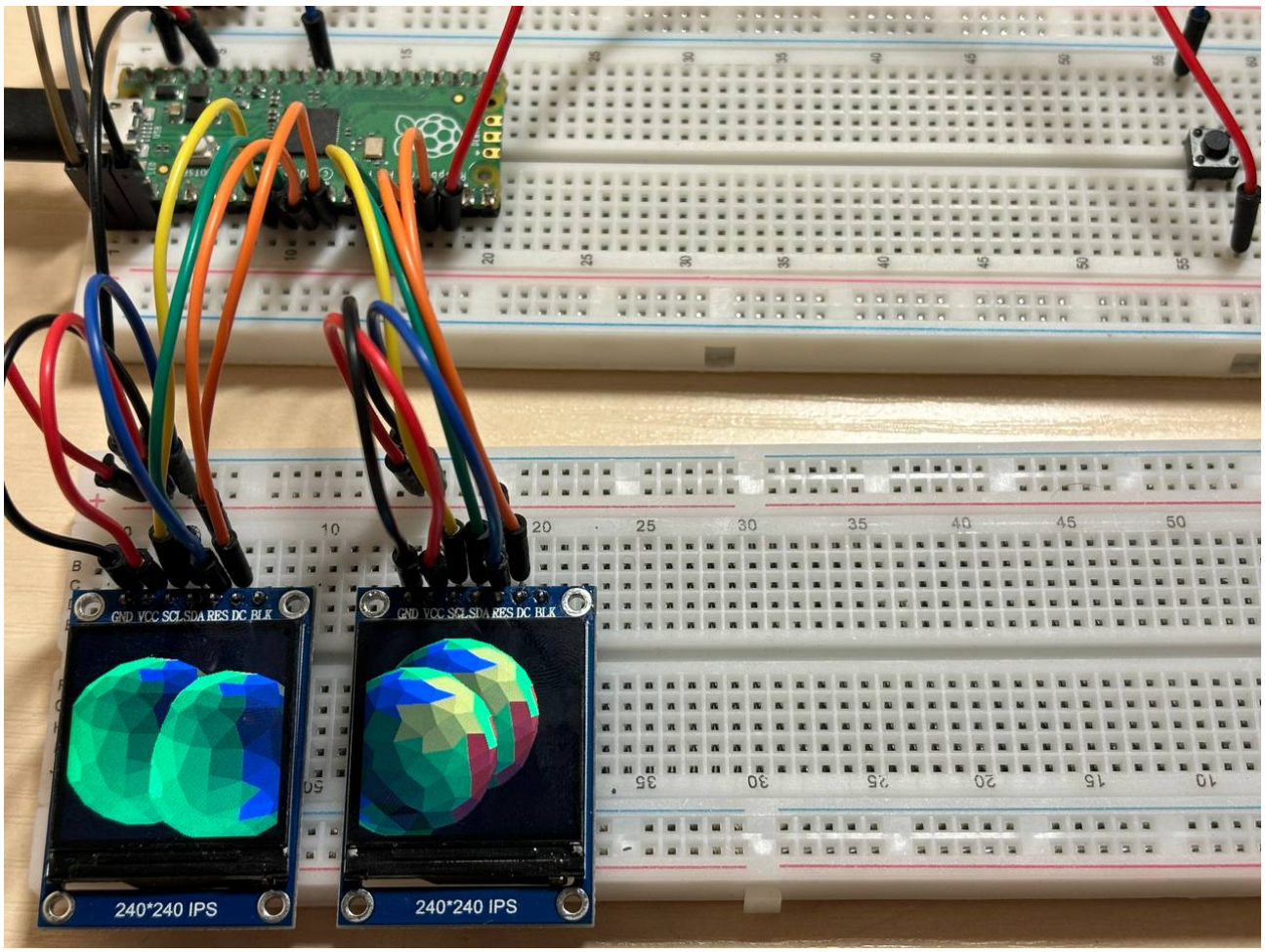

Рисунок 10 – Пример работы программно-аппаратного комплекса

На рисунке 11 представлен пример работы с пользовательским интерфейсом программы.

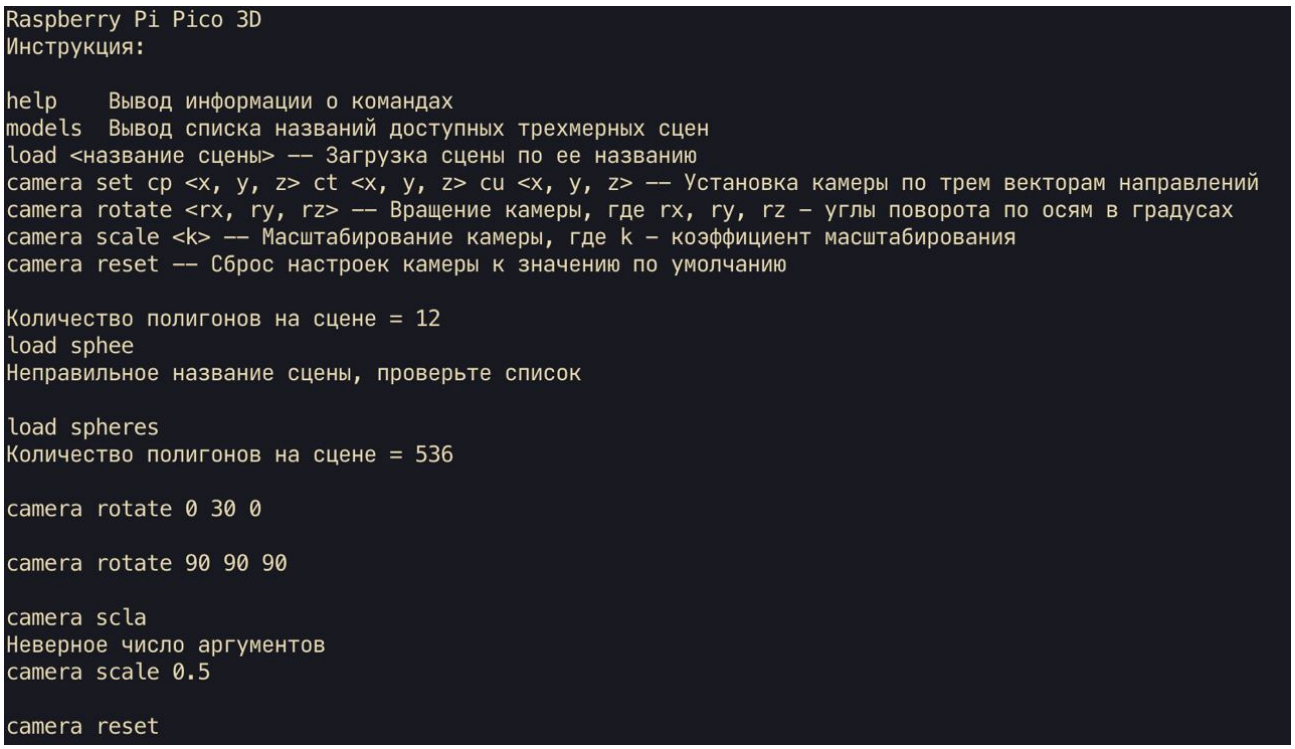

Рисунок 11 – Пример пользовательского интерфейса

3.8 Вывод из технологической части

В данном разделе были рассмотрены детали реализации программно-аппаратного комплекса, описанного в конструкторской части работы, и приведены примеры работы макета устройства, а также пример работы с пользовательским интерфейсом.

4 Исследовательская часть

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Д. Р.* Алгоритмические основы машинной графики //. Т. 512. — Мир, 1989. — Гл. 4.
2. Удаление скрытых линий и поверхностей [Электронный ресурс]. — Режим доступа: http://algolist.manual.ru/graphics/delinvis.php (дата обращения: 10.7.2023).
3. Что такое Ray Tracing или трассировка лучей [Электронный ресурс]. — Режим доступа: https://club.dns-shop.ru/blog/t-99-videokartyi/30460-chto-takoe-ray-tracing-ili-trassirovka-luchei-chto-daet-i-kak-rabo (дата обращения: 25.9.2023).
4. Phong B. T. Illumination for Computer Generated Pictures //. — 1975.
5. STM32 32-bit Arm Cortex MCUs [Электронный ресурс]. — Режим доступа: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html (дата обращения: 1.10.2023).
6. Cortex-M0 [Электронный ресурс]. — Режим доступа: https://developer.arm.com/Processors/Cortex-M0 (дата обращения: 1.10.2023).
7. Cortex-M3 [Электронный ресурс]. — Режим доступа: https://developer.arm.com/Processors/Cortex-M3 (дата обращения: 1.10.2023).
8. Cortex-M4 [Электронный ресурс]. — Режим доступа: https://developer.arm.com/Processors/Cortex-M4 (дата обращения: 1.10.2023).
9. Cortex-M7 [Электронный ресурс]. — Режим доступа: https://developer.arm.com/Processors/Cortex-M7 (дата обращения: 1.10.2023).
10. ESP32 [Электронный ресурс]. — Режим доступа: https://www.espressif.com/en/products/socs/esp32 (дата обращения: 1.10.2023).
11. Xtensa LX6 Customizable DPU [Электронный ресурс]. — Режим доступа: https://mirrobo.ru/wp-content/uploads/2016/11/Cadence_Tensillica_Xtensa_LX6_ds.pdf (дата обращения: 1.10.2023).
12. Rspberry Pi [Электронный ресурс]. — Режим доступа: https://www.raspberrypi.org (дата обращения: 1.10.2023).

13. ARMv6-M Architecture Reference Manual [Электронный ресурс]. — Режим доступа: https://developer.arm.com/documentation/ddi0419/c/ (дата обращения: 1.10.2023).

14. ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition [Электронный ресурс]. — Режим доступа: https://developer.arm.com/documentation/ddi0406/latest/ (дата обращения: 1.10.2023).

15. Cortex-A [Электронный ресурс]. — Режим доступа: https://embeddedartistry.com/fieldmanual-terms/cortex-a (дата обращения: 1.10.2023).

16. Raspberry Pi Documentation [Электронный ресурс]. — Режим доступа: https://www.raspberrypi.com/documentation/microcontrollers/rp2040.html (дата обращения: 1.10.2023).

17. NXP Semiconductors [Электронный ресурс]. — Режим доступа: https://www.nxp.com (дата обращения: 1.10.2023).

18. *Süsstrunk S.*, *Buckley R.*, *Swen S.* Standard RGB Color Spaces //. T. 7. — Proc. IS&T/SID 7th Color Imaging Conference, 1999.

19. Camera/View space [Электронный ресурс]. — Режим доступа: https://learnopengl.com/Getting-started/Camera (дата обращения: 13.10.2023).

20. Perspective Projection [Электронный ресурс]. — Режим доступа: https://ogldev.org/www/tutorial12/tutorial12.html (дата обращения: 13.10.2023).

21. Coordinate Systems [Электронный ресурс]. — Режим доступа: https://learnopengl.com/Getting-started/Coordinate-Systems (дата обращения: 13.10.2023).

22. Serial Peripheral Interface (SPI) [Электронный ресурс]. — Режим доступа: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/chip-select-cs (дата обращения: 10.12.2023).

23. Методология IDEF0 [Электронный ресурс]. — Режим доступа: https://studme.org/87184/ekonomika/metodologiya_idef0 (дата обращения: 15.7.2023).

24. Raspberry Pi DMA programming [Электронный ресурс]. — Режим доступа: https://iosoft.blog/2020/05/25/raspberry-pi-dma-programming/ (дата обращения: 10.12.2023).

25. Документация по Microsoft C++ [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170&viewFallbackFrom=vs-2017 (дата обращения: 10.12.2023).

26. Introduction to the ARM Serial Wire Debug (SWD) [Электронный ресурс]. — Режим доступа: https://developer.arm.com/documentation/ihi0031/a/The-Serial-Wire-Debug-Port--SW-DP-/Introduction-to-the-ARM-Serial-Wire-Debug--SWD--protocol (дата обращения: 10.12.2023).

27. Open On-Chip Debugger [Электронный ресурс]. — Режим доступа: https://openocd.org/ (дата обращения: 10.12.2023).

28. Wavefront OBJ File Format [Электронный ресурс]. — Режим доступа: https://www.loc.gov/preservation/digital/formats/fdd/fdd000507.shtml (дата обращения: 10.12.2023).

29. Wavefront Material Template Library (MTL) File Format [Электронный ресурс]. — Режим доступа: https://www.loc.gov/preservation/digital/formats/fdd/fdd000508.shtml (дата обращения: 10.12.2023).