МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____ 2020 г.

Разработка демонстрационного комплекса для алгоритмов загораживания Робертса, Аппеля, Z-буфера

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____ Е.С. Ярош
«___»_____ 2020 г.

Автор работы,
студент группы КЭ-405
_____ К.А. Росс
«___»_____ 2020 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____ 2020 г.

Челябинск-2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____2020 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Росс Климу Александровичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы:** «Разработка демонстрационного комплекса для алгоритмов загораживания Робертса, Аппеля, Z-буфера» утверждена приказом по университету от 24 апреля 2020 г. №627

2. **Срок сдачи студентом законченной работы:** 1 июня 2020 г.

3. **Исходные данные к работе:**

   Обеспечить основной функционал приложения:

– предоставление теоретической информации по алгоритмам загораживания Робертса, Аппеля, Z-буфера;
– детальная визуализация работы алгоритмов;

– настройка параметров визуализации, в том числе выбор объекта визуализации, изменение скорости визуализации, перемещение между этапами хода работы алгоритмов.

4. **Перечень подлежащих разработке вопросов:**

– подробное изучение алгоритмов загораживания;
– анализ существующих аналогов;
– выбор инструментов для разработки системы;
– проектирование архитектуры системы;
– разработка собственного программного комплекса;
– оценка работоспособности разработанного программного комплекса.

5. **Дата выдачи задания:** 1 декабря 2019 г.

Руководитель работы _____/*Е.С. Ярош*/

Студент _____/*К.А. Росс*/

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	01.03.2020	
Разработка модели, проектирование	01.04.2020	
Реализация системы	01.05.2020	
Тестирование, отладка, эксперименты	15.05.2020	
Компоновка текста работы и сдача на нормоконтроль	24.05.2020	
Подготовка презентации и доклада	30.05.2020	

Руководитель работы _____/*Е.С. Ярош*/

Студент _____/*К.А. Росс*/

Аннотация

К.А. Росс. Разработка демонстрационного комплекса для алгоритмов загораживания Робертса, Аппеля, Z-буфера. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2020, 159 с., 54 ил., библиогр. список – 21 наим.

В рамках выпускной квалификационной работы производится разработка программного комплекса, предназначенного для изучения алгоритмов удаления невидимых линий и поверхностей, которые также называют алгоритмами загораживания.

После анализа существующей информации по алгоритмам загораживания были сделаны выводы, что имеется достаточно подробное теоретическое описание основных алгоритмов, но без наглядного представления их работы. В связи с этим была выявлена необходимость в создании такого ресурса, который помимо теоретического описания мог бы предоставлять демонстрацию работы этих алгоритмов.

Результатом работы является демонстрационный комплекс, который включает в себя теоретическую основу, блок-схемы и наглядное представление хода работы алгоритмов Робертса, Аппеля, Z-буфера. Комплекс программ направлен на понимание студентами методов и алгоритмов удаления невидимых линий и поверхностей – одного из самых сложных разделов компьютерной графики.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Компьютерная графика является одной из основных областей информационных технологий, которая включает в себя различные аспекты формирования изображений на компьютере. Конечным результатом применения средств компьютерной графики является изображение различного вида и назначения – от простых чертежей до реалистических образов естественных объектов. Такое разнообразие обосновано в первую очередь тем, что эти изображения используются практически в каждой научной или инженерной дисциплине, где важны наглядность восприятия информации.

Знание основ того, каким образом строится изображение на компьютере, необходимо каждому специалисту, связанному с IT-сферой. Это позволяет осуществлять разработку, осваивать современные графические системы. Поэтому изучение основ компьютерной графики является обязательным компонентом учебных программ по информационным направлениям.

В то же время современная компьютерная графика – сложная и разнообразная научно-техническая дисциплина. Большинство алгоритмов компьютерной графики к настоящему времени имеют под собой достаточно подробную теоретическую основу с использованием математического аппарата. Это, наряду с отсутствием достаточного количества примеров демонстрации применения, делает их затруднительными в изучении для большинства студентов. Так как одним из методов повышения эффективности обучения является наглядность, имеет смысл визуализировать работу отдельных алгоритмов с помощью технологий компьютерного моделирования.

В настоящее время нет единого комплекса примеров демонстрации алгоритмов, существуют лишь отдельные файлы, написанные на различных

языках программирования, для разных операционных систем. Данные файлы необходимо искать, скачивать, на это тратится время, причем не гарантируется безопасность файлов, корректность алгоритмов. Также файлы могут оказаться и вовсе неработоспособными.

Этими обстоятельствами обоснована необходимость разработки демонстрационного программного комплекса для изучения алгоритмов представления графической информации.

В рамках дипломной работы охватить все эти алгоритмы невозможно. Поэтому, в соответствии с заданием, будет разработан программный комплекс по методам и алгоритмам удаления невидимых линий и поверхностей. В проект кроме визуализации алгоритмов входит теоретический материал по ним, блок-схемы. Теоретический материал взят из учебно-методического пособия «Методы и средства представления графической информации» автора к.т.н. Ярош Е.С., алгоритмы реализованы по этому же пособию [1].

Целью выпускной квалификационной работы является разработка демонстрационного комплекса программ для изучения следующих алгоритмов загораживания: Робертса, Аппеля, Z-буфера.

Для достижения поставленной цели, необходимо решить следующие задачи:

1. Рассмотреть алгоритмы загораживания, указанные в задании.
2. Сделать обзор существующих аналогов, демонстрирующих работу алгоритмов.
3. Выбрать инструменты для разработки системы.
4. Разработать архитектуру собственного программного комплекса.
5. Выполнить программную реализацию программного комплекса.
6. Оценить работоспособность готового программного комплекса.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Задачу удаления невидимых линий и поверхностей называют также задачей загораживания. Она возникает при построении реалистических трехмерных изображений и является весьма сложной. Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства.

Проиллюстрируем необходимость удаления невидимых линий ситуацией, изображенной на рисунке 1.1. Рисунок наглядно демонстрирует, что изображение без удаления невидимых линий воспринимается неоднозначно.

На рисунке 1.1 (а) приведен каркасный чертеж куба, т.е. трехмерный объект в виде изображения его ребер. Иллюстрацию можно интерпретировать двояко: либо как вид куба сверху слева (б), либо снизу справа (в). Применение одного из алгоритмов загораживания позволяет избавиться от неоднозначности.

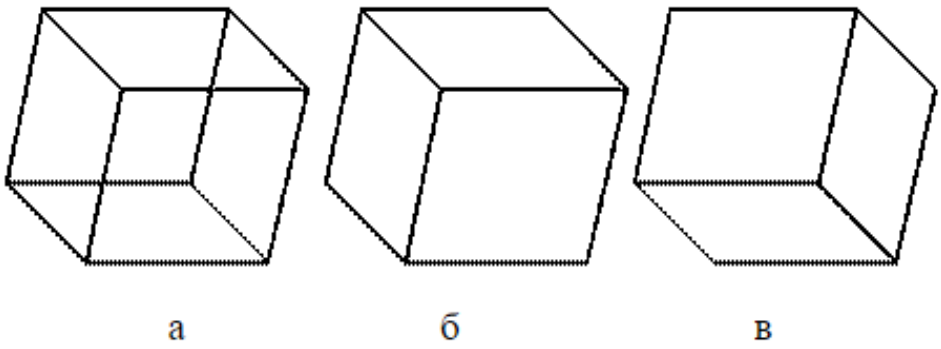


Рисунок 1.1 – Необходимость удаления невидимых линий

Таким образом, цель применения алгоритма заключается в определении того, что видно из некоторой точки картинной плоскости. Несмотря на простоту

формулировки, эта задача является достаточно сложной и может требовать больших вычислительных ресурсов.

Сложность задачи удаления невидимых линий и поверхностей привела к появлению различных способов ее решения. Методы решения задачи удаления невидимых объектов можно разделить [2]:

- по способу представления объектов: векторно-полигональные модели, сеточные модели и т.п.; большинство рассматриваемых методов работает с наиболее распространенным полигональным представлением;
- по способу визуализации сцены: различают каркасное и сплошное изображение трехмерных тел; в первом случае рисуются только ребра соответствующего каркаса, и задача удаления относится к невидимым линиям (ребрам); во втором случае рисуются закрашенные грани, и задача удаления касается невидимых поверхностей (граней);
- по пространству, в котором проводится анализ видимости объектов: можно решать задачу удаления невидимых объектов в исходном трехмерном пространстве объектов (с координатами x, y, z) или на картинной плоскости, работая с проекциями объектов;
- по виду (точности) получаемых результатов: различаются непрерывные, дискретные и смешанные методы; первые вырабатывают результат в виде набора областей (отрезков) в объектном пространстве, считающихся видимыми и заданных с машинной точностью (то есть не привязанных к растру картинной плоскости); вторые производят информацию о ближайшем объекте для любого пикселя экрана, давая приближенное решение задачи видимости (только для узлов растровой решетки); третьи выполняют часть работы в объектном пространстве (как непрерывные методы), а часть – в картинной плоскости (как дискретные методы).

Кроме того, методы могут отличаться способом сортировки объектов, который часто встречается в задачах загораживания.

Очевидно, что в рамках дипломной работы охватить абсолютно все алгоритмы загораживания невозможно. Поэтому данная работа ограничивается алгоритмами Робертса, Аппеля, Z-буфера – одними из самых популярных алгоритмов этого класса.

Для того, чтобы сделать обоснованный обзор-анализ существующих разработок, сформулируем минимальные требования, которыми должна обладать программная система:

1. Наглядная демонстрация работы одного или нескольких алгоритмов загораживания.
2. Применение алгоритмов для полигональных объектов с гранями в виде выпуклых многоугольников, так как это ограничение накладывается алгоритмом Робертса.

1.2. ОБЗОР АНАЛОГОВ

Поиск программных комплексов для демонстрации всех требуемых алгоритмов оказался безрезультатным. Были найдены примеры демонстрации отдельных алгоритмов, написанные на различных языках программирования. Общий их недостаток заключается в том, что пользователям предоставляются уже отредактированные объекты, без визуализации самого процесса применения алгоритма. В некоторых случаях вся демонстрация ограничивается одним или несколькими изображениями.

Для ознакомления с некоторыми из найденных ресурсов необходимо скачивать исполняемые файлы. Их можно использовать только после проверки их файлов на безопасность, работоспособность и корректность алгоритма.

В таблице 1.1 приведены наиболее соответствующие требованиям решения, найденные в Интернете, их достоинства и недостатки. На рисунках 1.2-1.4 представлены скриншоты каждого из примеров.

Таблица 1.1 – Преимущества и недостатки имеющихся решений

Наименование	**Преимущества**	**Недостатки**
1. Теоретические материалы [3-6]	– описывают все методы и алгоритмы удаления невидимых линий; – содержат подробное теоретическое описание алгоритмов.	– отсутствует визуализация применения.
2. Программный пакет на языке C++ (содержит Алгоритм Робертса) [7]	– охватывает не только алгоритмы загораживания, но и другие алгоритмы компьютерной графики; – содержит визуальное представление; – совместим с различными устройствами.	– отсутствуют настройки визуализации; – отсутствует теоретическое обоснование; – применение только для одного объекта в сцене.
3. Алгоритм Z-буфера с применением технологии WebGL [8]	– содержит визуальное представление; – визуальное представление с применением 3D-моделирования; – возможность посмотреть программный код; – совместимость с различными устройствами.	– отсутствует поэтапная визуализация алгоритма; – отсутствует теоретическое обоснование.
4. Программный пакет на языке Java [9]	– визуальное представление с возможностью настройки визуализации; – есть возможность посмотреть исходный программный код; – совместим с различными устройствами.	– отсутствует поэтапная визуализация алгоритма; – отсутствует теоретическое обоснование.

Найденные решения отличаются разнообразием инструментов разработки, без привязки к конкретным технологиям. Разработчик выбирает их на свое усмотрение исходя из установленных требований.

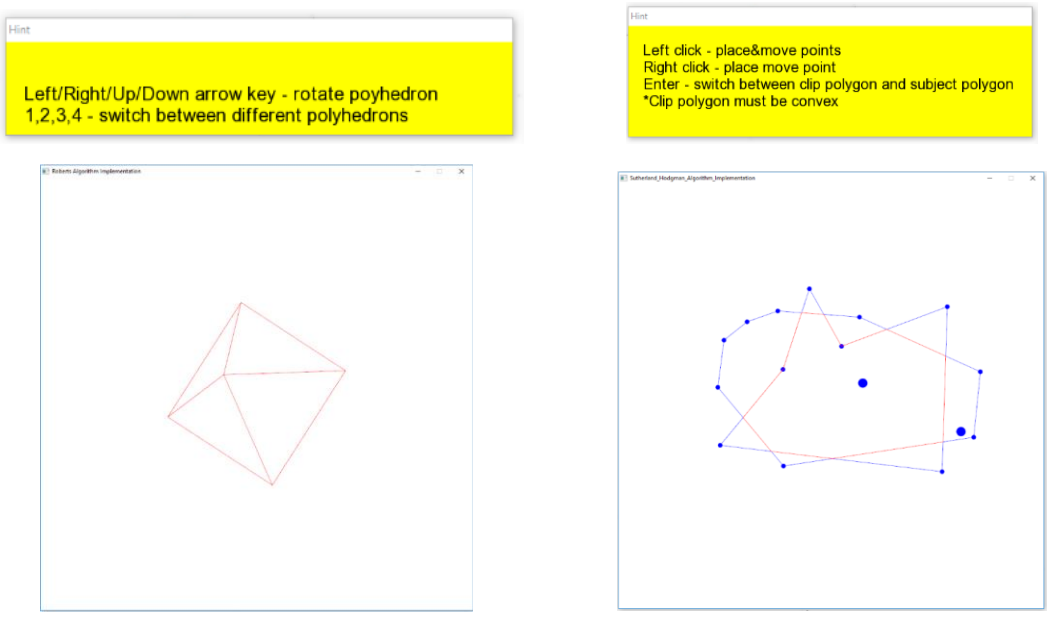


Рисунок 1.2 – Программный пакет на языке C++

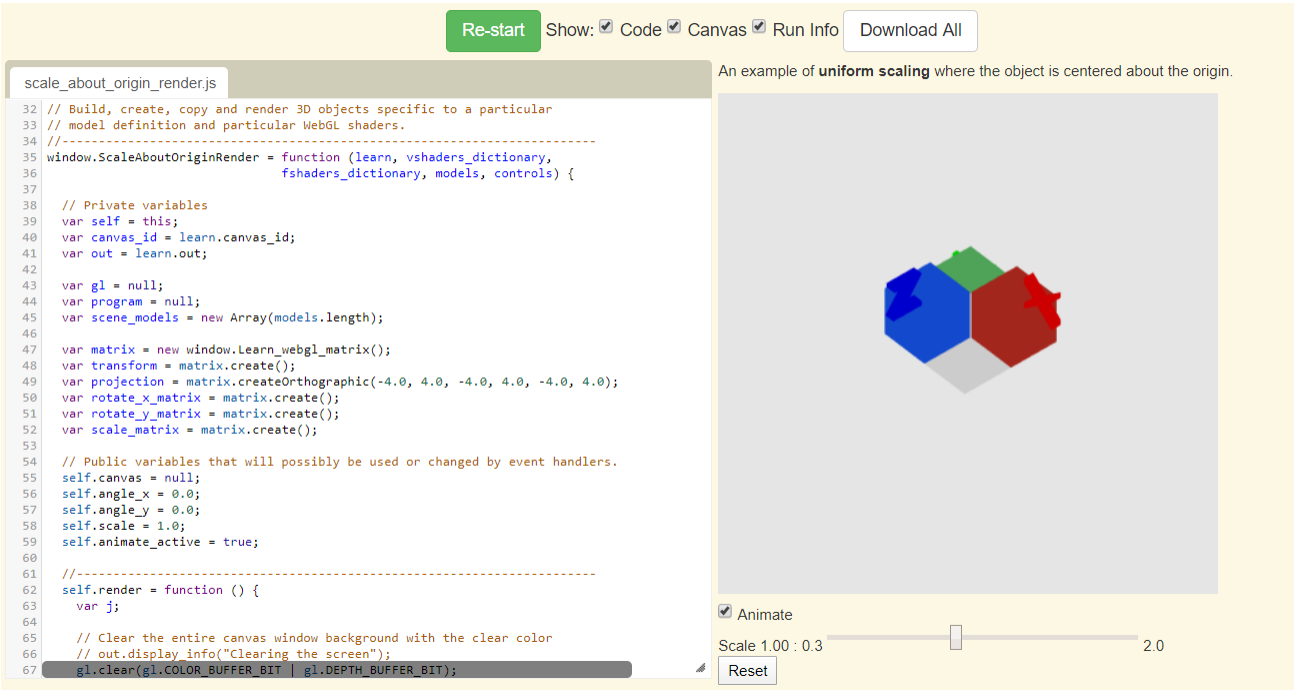


Рисунок 1.3 – Алгоритм Z-буфера с применением технологии WebGL

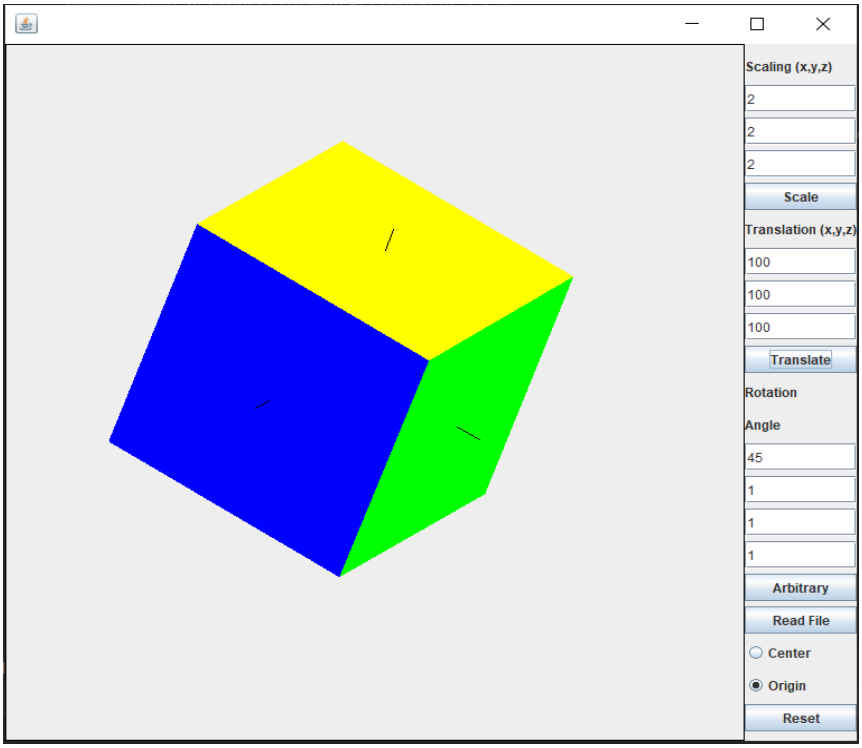


Рисунок 1.4 – Программный пакет на языке Java

1.3. РАЗРАБОТКА НОВОГО РЕШЕНИЯ

Разработка собственного демонстрационного программного комплекса (далее ДПК), соответствующего требованиям из п. 1.1, повысит эффективность изучения алгоритмов загораживания, удобство проведения занятий по компьютерной графике.

Учитывая недостатки имеющихся решений, следует включить в ДПК:

– теоретический раздел, содержащий, учебный материал и блок-схемы по каждому из алгоритмов, указанных в задании;
– демонстрацию работы алгоритмов с детальным рассмотрением того, какая часть изображения привела к удалению того или иного ребра;
– возможность производить настройку демонстрации, исходя из особенностей каждого алгоритма;

– возможность манипулирования процессом демонстрации, а именно постановки на паузу, перемещения между этапами алгоритмов.

Для теоретического раздела материал следует взять из учебно-методического пособия автора, к.т.н. Ярош Е.С. «Методы и средства представления графической информации». Блок-схемы алгоритмов построить по вышеупомянутому пособию.

Правильный выбор инструментов разработки обеспечит баланс между удобством разработки и производительностью готового продукта. После выбора инструментов следуют этапы:

– проектирование архитектуры ДПК;
– разработка программного кода алгоритмов;
– настройка графического API для визуализации хода работы алгоритмов;
– создание пользовательского интерфейса;
– создание теоретического раздела и блок-схем;
– оценка работоспособности ДПК.

Преимущества нового ДПК перед найденными решениями из пункта 1.2:

– алгоритмы располагаются в одном месте, что обеспечивает удобство и исключает затраты на поиск;
– гарантия работоспособности и безопасности приложения;
– предоставление настроек визуализации: ввод исходных данных, изменение скорости визуализации, изменение положения точки наблюдения;
– наличие раздела с теоретическим материалом, содержащим блок-схемы.

Из достоинств комплекса вытекает и его недостаток – требуется больше памяти по сравнению с отдельно взятыми примерами, следовательно, он менее компактен.

2. ПЛАНИРОВАНИЕ

2.1. ОСНОВНЫЕ ТРЕБОВАНИЯ

Для отражения предоставляемого функционала системы для разных групп пользователей строится диаграмма прецедентов (вариантов использования). Основными группами пользователей разрабатываемого проекта являются:

- преподаватели компьютерной графики, которые могут использовать ДПК для проведения лекций/практических занятий по темам, затрагивающим изучение алгоритмов загораживания;
- студенты, которые изучают эти алгоритмы на определенных дисциплинах;
- другие люди, заинтересованные в данной предметной области.

В ДПК функционал для перечисленных групп пользователей идентичен, поэтому в диаграмме прецедентов достаточно двух акторов – непосредственно пользователя программы и графического API. На диаграмме использования (рисунок 2.1) отражены сценарии работы с системой, которые приводят к желаемому результату (АУНЛ – алгоритмы удаления невидимых линий).

После запуска программы пользователю системы открывается стартовое меню с выбором раздела для изучения. Это могут быть отдельные алгоритмы или иные разделы с дополнительно информацией.

После выбора алгоритма для изучения пользователь может ознакомиться с теоретической информацией, которая включает блок-схему алгоритма, и затем перейти к процессу визуализации.

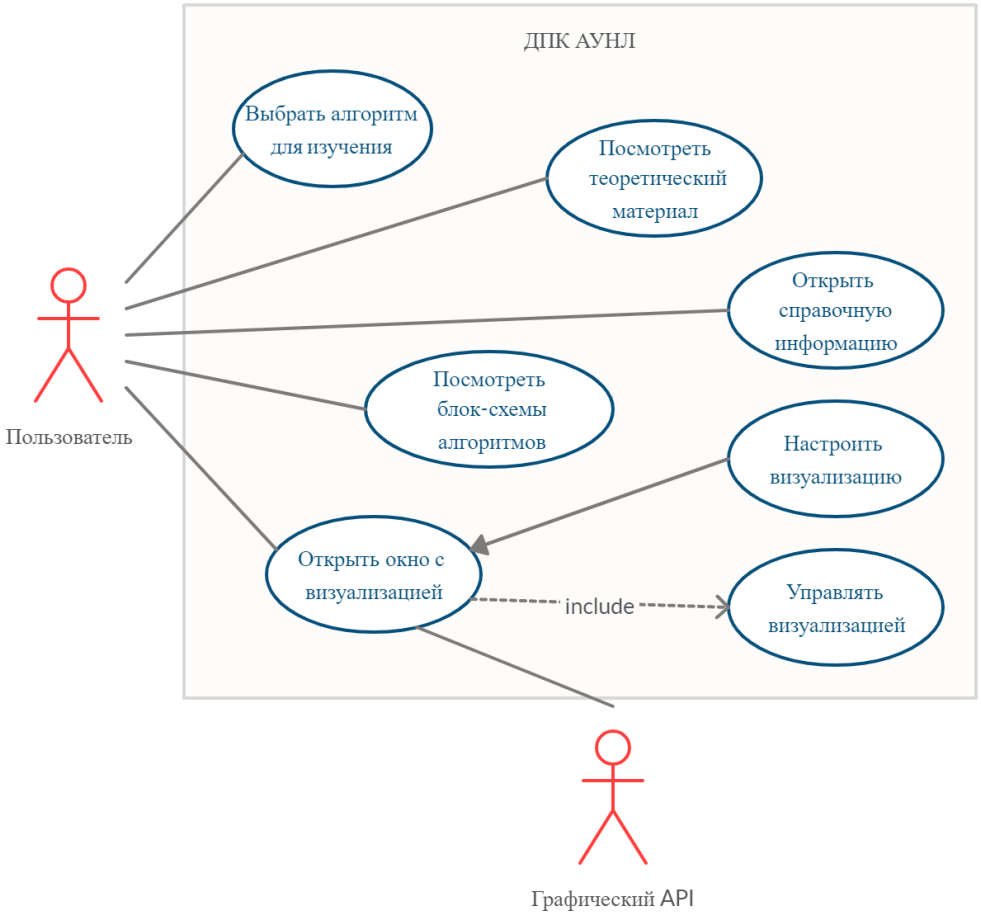


Рисунок 2.1 – Диаграмма вариантов использования системы

Также пользователь может настраивать процесс визуализации, в том числе управлять скоростью отрисовки, определять положение точки наблюдения, выбирать объект, для которого будет применяться алгоритм. Пользователю также доступна функция управление процессом визуализации, а именно переход от одного этапа алгоритма к другому.

2.2. ПЛАТФОРМА

Одно из требований заказчика – возможность поддержки платформы Windows 7x32 и выше. В настоящий момент Windows является самой

распространенной операционной системой [11], которая установлена на компьютерах в большинстве учебных заведений. Одним из преимуществ 32-х битной архитектуры Windows является то, что программы, написанные для 64-разрядной системы, не будут работать на 32-х разрядной, в то же время ПО под x32 работает как на 32, так и на 64-битной системе.

2.3. АНАЛИЗ И ВЫБОР СРЕДСТВ РАЗРАБОТКИ

Для создания программного комплекса возможными средствами разработки могут являться различные языки программирования с поддержкой графических библиотек. ДПК будет представлен в виде desktop-приложения для Windows, из языков программирования для подобных проектов самыми распространенными являются C, C++, C#, Java.

После небольшого анализа было решено, что наиболее подходящим для данной работы является язык программирования C#. Выбор между C++ и Java делался бы при условии организации кроссплатформенного приложения. Преимущество C# перед C – нацеленность на объектно-ориентированное программирование, что удобнее для разработки объемных программ. Также стоит отметить, что из-за простоты языка C# разработка на нем идет быстрее, чем на C++. Из недостатков можно отметить относительно невысокую производительность (медленнее, чем язык C, но сравним с Java). Так как в ДПК не требуется очень быстрых вычислений, то недостаток не является значимым.

После выбора языка C# в качестве языка программирования наиболее рациональной для разработки средой является Visual Studio 2017 Community Edition. Это бесплатный продукт компании Microsoft, который предоставляет мощный и удобный инструментарий для разработки. С помощью данной среды помимо консольных приложений можно разрабатывать приложения с

графическим интерфейсом, что является одной из ключевых особенностей программного комплекса. В Visual Studio присутствует возможность подключения сторонних библиотек для расширения функционала. Отладчик позволяет найти ошибки в исходном коде программы, посмотреть, как изменяются значения необходимых переменных, тем самым предоставляя разработчику информацию для контроля правильности вычислений.

В качестве интерфейса программирования приложения была выбрана технология Windows Forms, которая является основным средством для разработки desktop-приложений под Windows. Данное средство упрощает доступ к элементам интерфейса Windows за счет создания обёртки для существующего Win32 API в управляемом коде, который отвечает за графический интерфейс пользователя.

Для работы с графикой в приоритете такие графические API, как OpenGL, DirectX, Vulkan [12]. В конечном счете выбрана библиотека OpenGL по следующим причинам:

- работа в ДПК идет лишь с графическими примитивами и не требует современных эффектов;
- технология OpenGL наиболее популярна среди встраиваемых графических библиотек, отсюда следует большое количество учебных пособий, примеров использования и поддержки сообщества;
- код, написанный с использованием OpenGL, выглядит нагляднее, чем с использованием DirectX или Vulcan, что проще для разработчика приложения.

Для совместимости с C# будет использоваться технология OpenTK [13]. OpenTK является графической библиотекой на языке C# и предоставляет доступ к графическим инструментам, содержащимся в OpenGL.

2.4. АНАЛИЗ И ВЫБОР ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

В программном комплексе используется трехмерная графика. Построение графических примитивов – ресурсозатратная операция, которая занимает весь основной поток приложения. Вдобавок для наибольшей наглядности пользователю необходимо видеть процесс визуализации алгоритма, который в свою очередь может занимать время. Из-за этого выполнение пользователем других операций параллельно с отрисовкой графики невозможно, так как на время выполнения задачи будет блокироваться графический интерфейс. Вследствие этого необходимо организовать многопоточное выполнение программы.

В языке C# классы, поддерживающие многопоточное программирование, определены в пространстве имен System.Threading, которое содержит различные типы, главным из которых является класс Thread (Поток). Функция рисования графики будет вынесена в отдельный поток приложения.

3. ПРОЕКТИРОВАНИЕ

Для реализации данной системы необходим следующий набор подсистем:

1. *Графическое ядро*. С помощью графического ядра обеспечивается обработка и отображение результатов алгоритмических вычислений на экран пользователя.
2. *Запрограммированные алгоритмы*. Каждый из реализованных алгоритмов принимает на вход данные об расположении объекта в пространстве, производит необходимые вычисления. Результаты на выходе передаются графическому ядру.
3. *Приложение с графическим пользовательским интерфейсом*. С помощью графического интерфейса обеспечивается общее управление системой, визуальное отображение информации по алгоритмам, в том числе теоретической информации в виде текста и визуализации работы с помощью графического ядра.

При описании архитектуры разрабатываемого программного обеспечения была спроектирована диаграмма компонентов, представленная на рисунке 3.1. Она отображает взаимодействия логических частей приложения.

Теоретический раздел и визуализатор – основные модули приложения. Каждый из них должен быть выполнен в отдельном окне Windows. Из одного модуля можно свободно перейти в другой с сохранением информации. Визуализатор в свою очередь содержит модули настройки визуализации и отображения буферов кадра и глубины (для алгоритма Z-буфера).

В приложении должно быть реализовано 5 окон:

– главное меню с выбором алгоритма для изучения;
– теоретический раздел;

– визуализатор;
– окно настроек визуализации;
– окно с отображением буферов кадра и глубины.

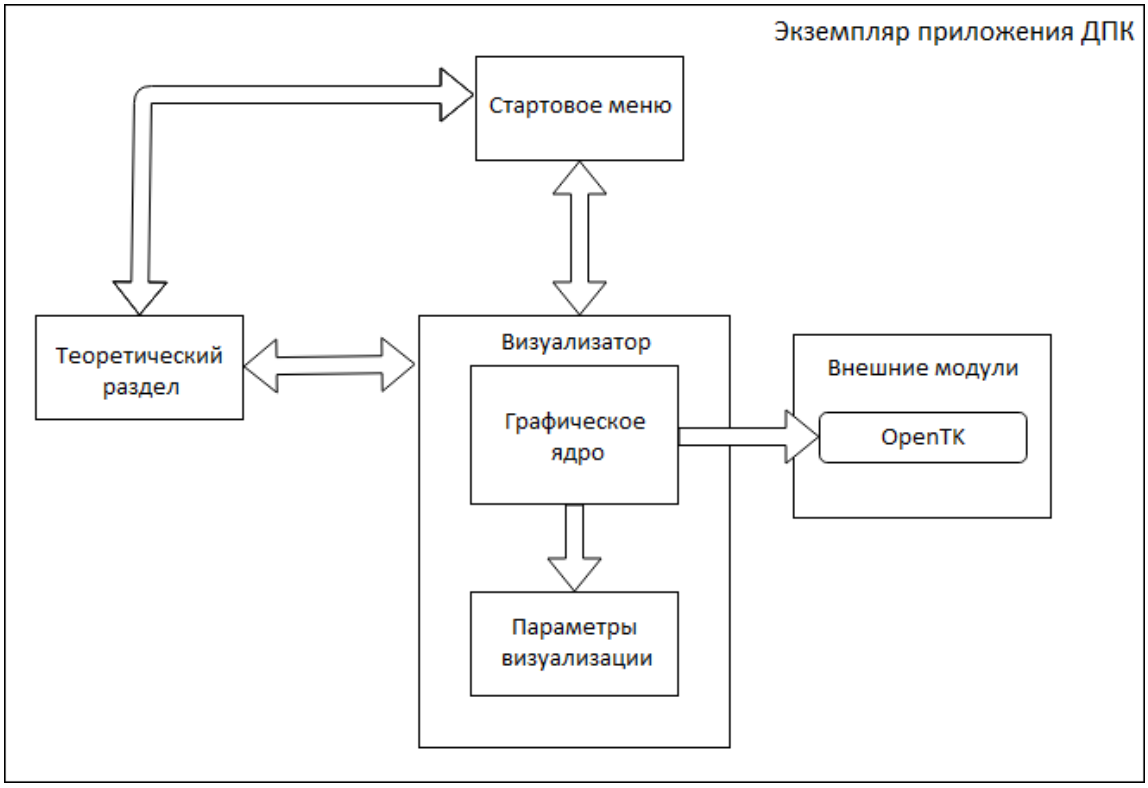


Рисунок 3.1 – Диаграмма компонентов проектируемой систем

При выборе в меню определенного раздела с алгоритмом аналогичный раздел должен выбираться в теоретическом разделе и в визуализаторе, если не было последующих изменений, иначе в качестве изучаемого должен устанавливаться последний выбранный алгоритм из верхнего меню.

Для функционирования графики в визуализатор входит внешняя библиотека OpenTK, предназначенная для использования OpenGL функций. Функции, реализуемые данной библиотекой, отвечают за системный уровень операций ввода-вывода при работе с операционной системой: создание окна, управление окном, мониторинг ввода с клавиатуры и событий мыши. В данном

приложении эти функции необходимы для создания ортографической проекции для трехмерного моделирования.

Для описания классов, методов, атрибутов системы и взаимосвязи между ними необходимо построить диаграммы классов UML. Будет использовано упрощенное представление класса, то есть выбираются только основные из его атрибутов. Стоит отметить, что диаграммы, построенные на этапе проектирования, могут устаревать в ходе разработки программной системы и терять свое значение при изменении проекта.

На рисунке 3.2 приведена диаграмма классов, предназначенных для программного представления пространственных объектов. Краткая характеристика этих классов:

- экземпляры класса «Вершина», представляют собой точку в пространстве. Переменные, содержащие в себе информацию о координатах, являются основными полями класса. Значением поля w является четвертая компонента, присущая однородным координатам;
- экземпляры класса «Ребро» хранят информацию о положении ребра в пространстве. Основные поля – координаты вершин начала и конца отрезка, дополнительные поля необходимы для задания свойств ребра;
- каждый экземпляр класса «Грань» хранит информацию об одной из граней многогранника. Здесь основным полем является список всех ребер, формирующих некую грань;
- объекты сцены представляют собой коллекцию (List) экземпляров класса «Многогранник», которые в свою очередь задаются списком граней.

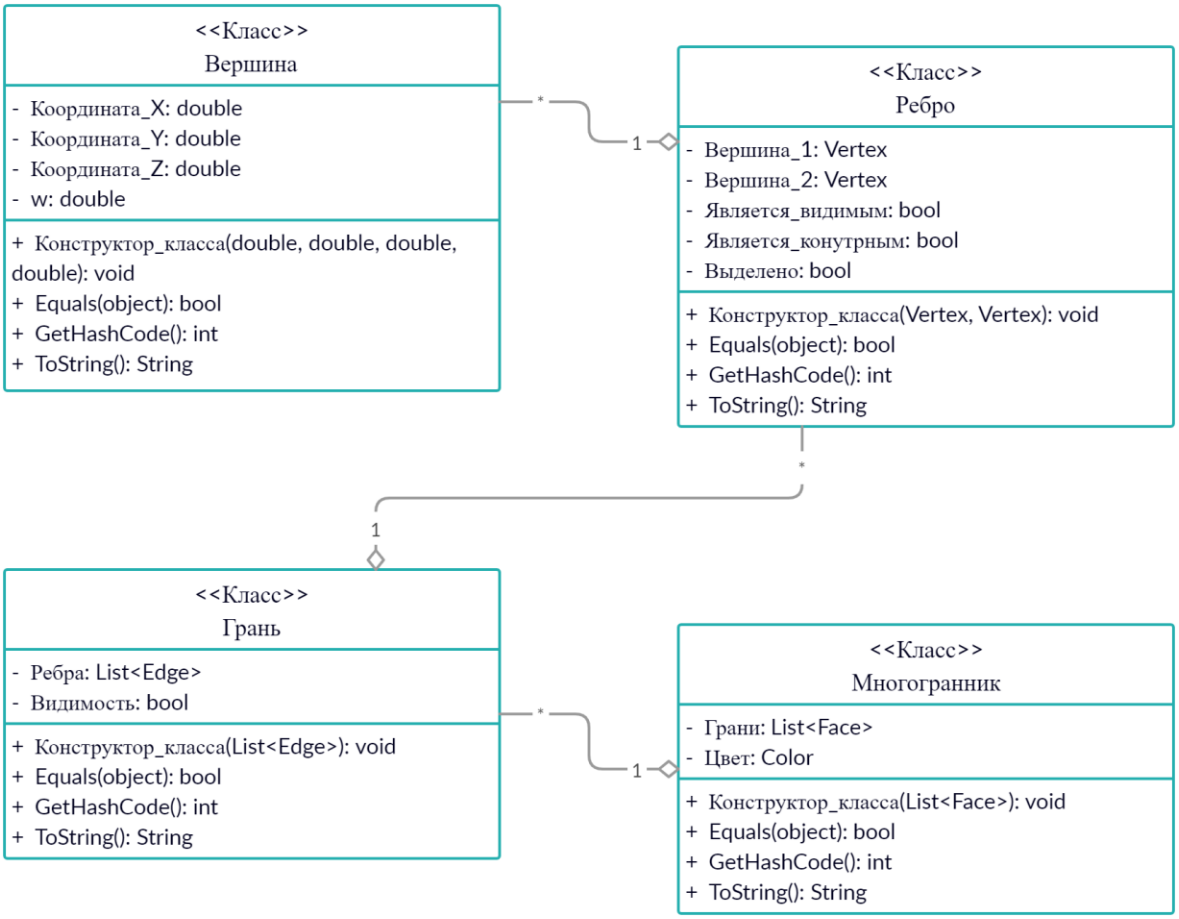


Рисунок 3.2 – Диаграмма классов представления объекта

На рисунке 3.3 приведена диаграмма классов алгоритмов загораживания. Абстрактный класс «Алгоритм» содержит определение основных методов, которые должны быть реализованы каждым из наследуемых классов:

– методы, предназначенные для непосредственной обработки сцены на предмет невидимых линий;
– методы, отвечающие за распределение хода работы алгоритма на этапы, перемещения между этими этапами.

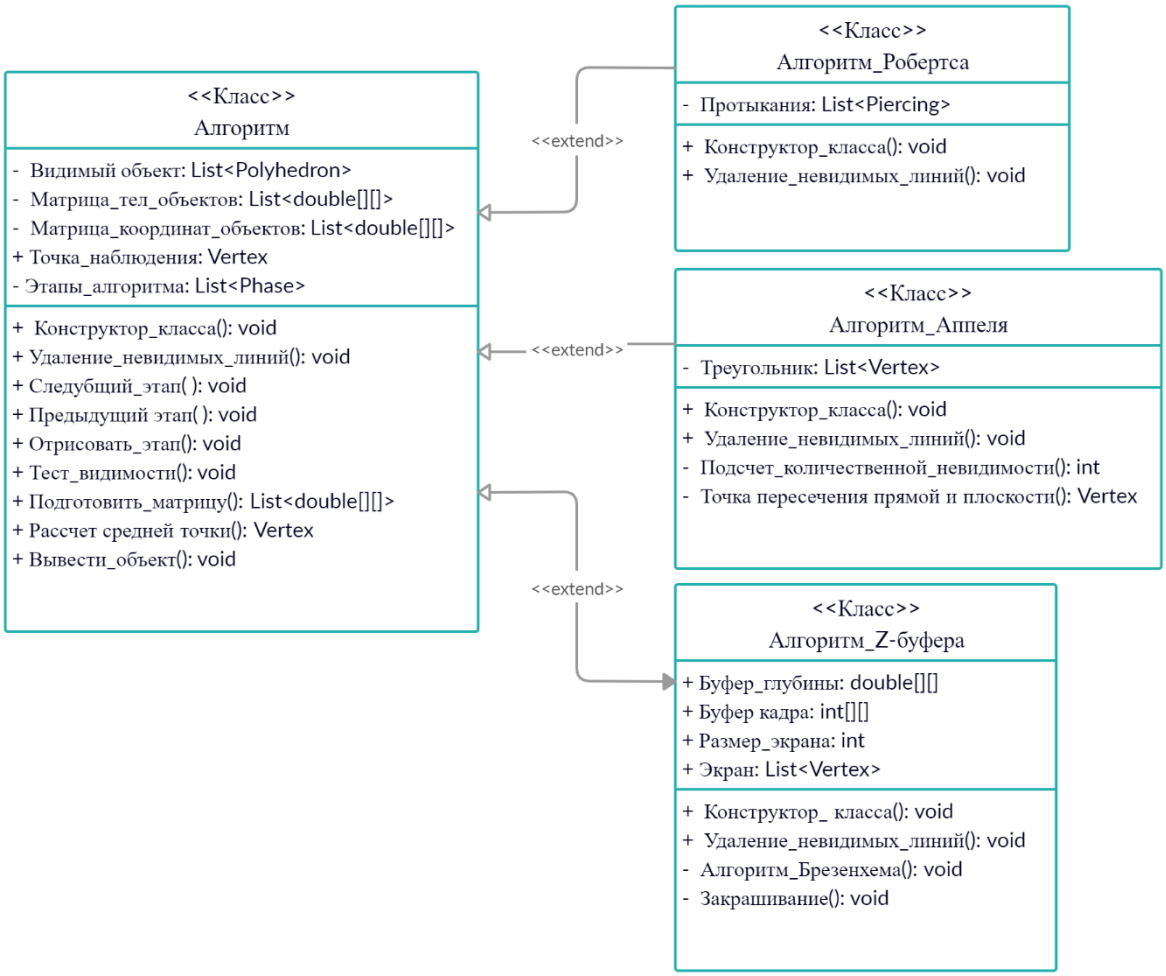


Рисунок 3.3 – Диаграмма классов алгоритмов

Для организации перемещения между этапами в каждом классе, реализующем алгоритм, будет определён внутренний класс «Этап» (рисунок 3.4). Экземпляры класса будут иметь атрибуты, каждый из которых описывает все свойства текущего этапа выбранного алгоритма. Например, для алгоритма Z-буфера, к полям, отраженным на рисунке 3.4, будут также добавлены поля для хранения буферов глубины и кадра.

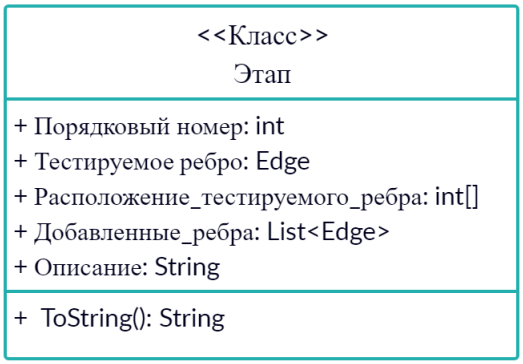


Рисунок 3.4 – Класс, представляющий некий этап хода работы алгоритмов

Для описания взаимосвязи между всеми классами была построена общая диаграмма классов (рисунок 3.5).

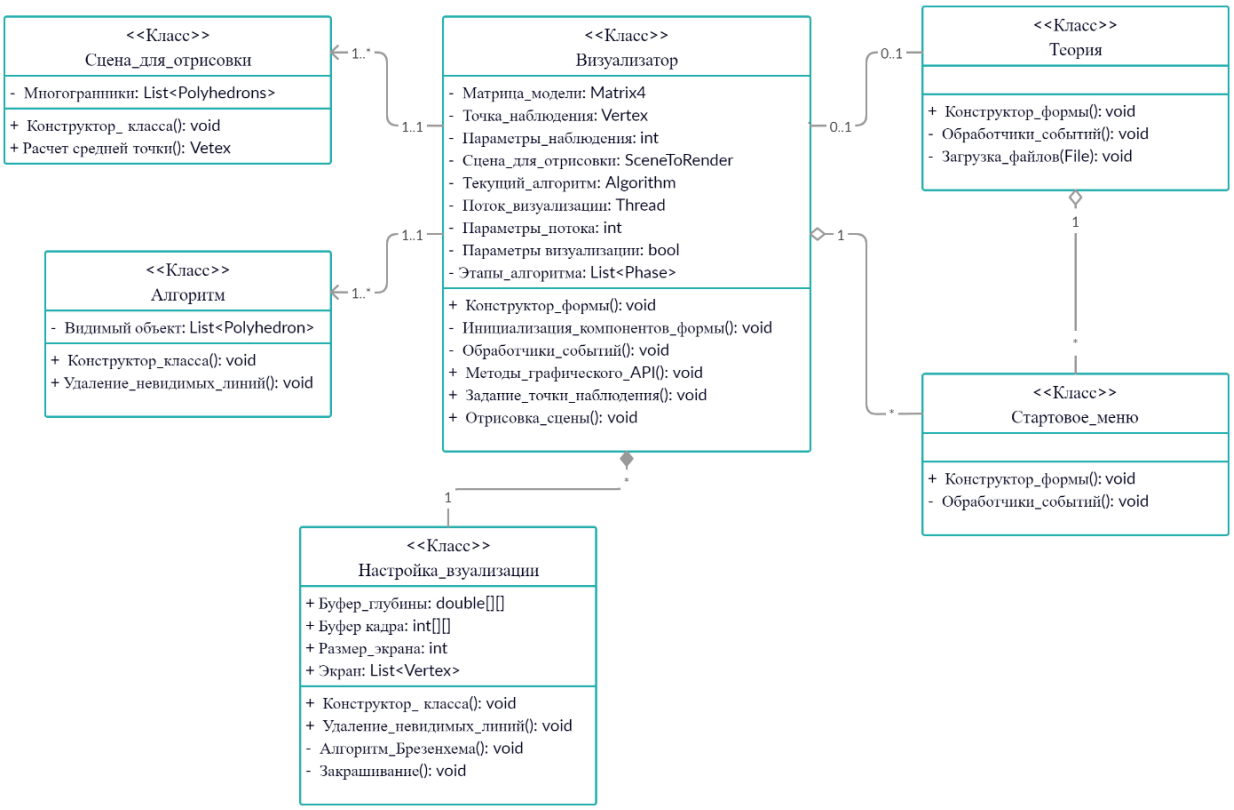


Рисунок 3.5 – Диаграмма компонентов проектируемой систем

В основе своей эти классы представляют формы Windows Forms – визуальные поверхности, на которые выводится информация для пользователя.

Главная форма «Визуализатор» предназначена для демонстрации применения алгоритмов. Основные ее атрибуты можно разделить на две категории:

– параметры отрисовки объектов с помощью графического API;
– параметры процесса визуализации, в том числе перемещения между этапами хода работы алгоритмов.

Основная зависимость между классами представляет собой ассоциацию – структурную связь между элементами, которая описывает набор связей, существующих между объектами. При ассоциации объекты одной сущности (класса) связаны с объектами другой сущности так, что можно перемещаться от объектов одного класса к другому. Взаимосвязи объектов классов на представленных диаграммах: агрегация и композиция.

Агрегация — разновидность ассоциации, представляющая структурную связь целого с его частями. Агрегация встречается, когда один класс является коллекцией или контейнером других. По умолчанию время существования классов не зависит от времени существования классов-контейнеров в которых они содержатся. Другими словами, если контейнер будет уничтожен, то его содержимое – нет. Например, классы «Меню» и «Теоретический материал», «Теоретический материал» и «Визуализация» связаны агрегацией;

Композиция является более строгим вариантом агрегации с четко выраженными отношениями владения. При композиции время существования экземпляров класса-контейнера имеет жёсткую зависимость от времени существования экземпляров содержащихся классов. Другими словами, если контейнер будет уничтожен, то всё его содержимое также будет уничтожено. Например, классы «Настройка визуализации» и «Визуализация» связаны композицией, т.е. если закрыть визуализатор, то нельзя будет увидеть настройки визуализации.

4. РЕАЛИЗАЦИЯ

Каждый из алгоритмов загораживания применяется к пространственным объектам. Для их отображения на экране пользователя используется технология OpenGL внутри обычной GUI-программы [14]. Графический элемент управления glControl накладывается на поверхность формы для последующей отрисовки графики. Под формой понимается видимая поверхность окна, включающая информацию для конечного пользователя, а также содержащая в себе набор элементов управления для работы с представленными данными или взаимодействия с пользователем.

Формальное описание того, как устроен процесс создания сцены в OpenGL:

1. Позиционирование объема видимости в пространстве, т.е. установка камеры в каких-либо координатах.
2. Установка в пространстве модели (объекта), которая будет попадать в объем видимости нашей камеры.
3. Проецирование модели, при котором определяется форма видимого объема в пространстве.
4. Получение изображение объекта в рамках окна просмотра.

В процессе создания сцены используется несколько координатных пространств (рисунок 4.1) по той причине, что некоторые операции становятся более понятными или более простыми в определенных координатных системах [15, 16].

Все объекты задаются в своих 3-х мерных координатах. Эти координаты называются объектными. Для того, чтобы построить сцену из разных 3D-объектов, необходимо объектные координаты каждого из объектов умножить на

модельную матрицу (model Matrix). Как итог получаются новые координаты объектов в новом общем “мировом” пространстве.

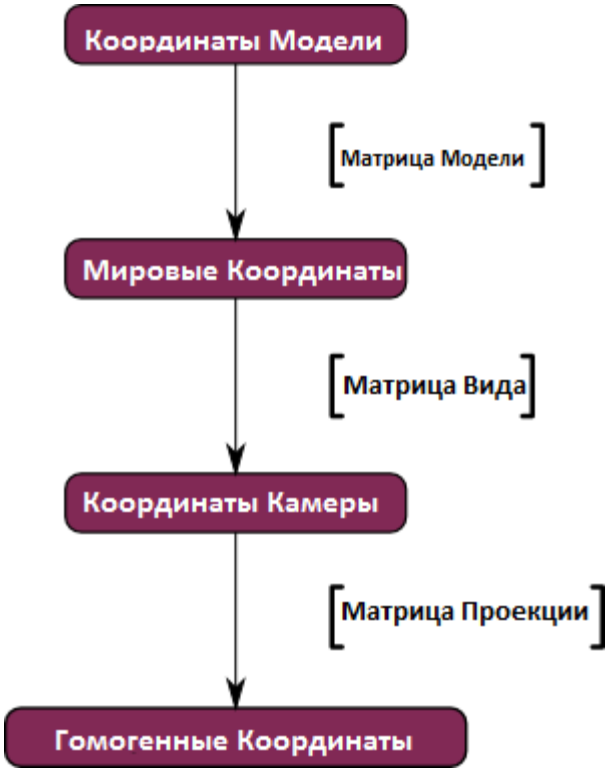


Рисунок 4.1 – Матричные преобразования

В то же время камера (точка наблюдения) может быть направлена на все объекты с разных сторон, перемещаться, приближаться и удаляться. В таком случае необходимо умножить координаты объектов мировом пространстве на матрицу видового преобразования (view Matrix). Как итог получаются видовые координаты каждого объекта.

В OpenGL матрица модельного преобразования (model Matrix) совмещена с матрицей видового преобразования (view Matrix) в одну (modelView Matrix). Отдалить объект в таком случае можно двумя способами:

– изменить мировые координаты объекта (т.е. отдалить сам объект);
– отдалить от объекта камеру (получить новые видовые координаты).

Следующим шагом координаты необходимо умножить на матрицу проекции (projection Matrix), которая либо отвечает за frustum – пирамиду охвата видимости, образуемую шестью плоскостями (рисунок 4.2). С помощью функции CreatePerspectiveField() можно создать матрицу перспективы на основе параметров поля зрения, пропорций и расстояния до ближней и дальней плоскости просмотра.

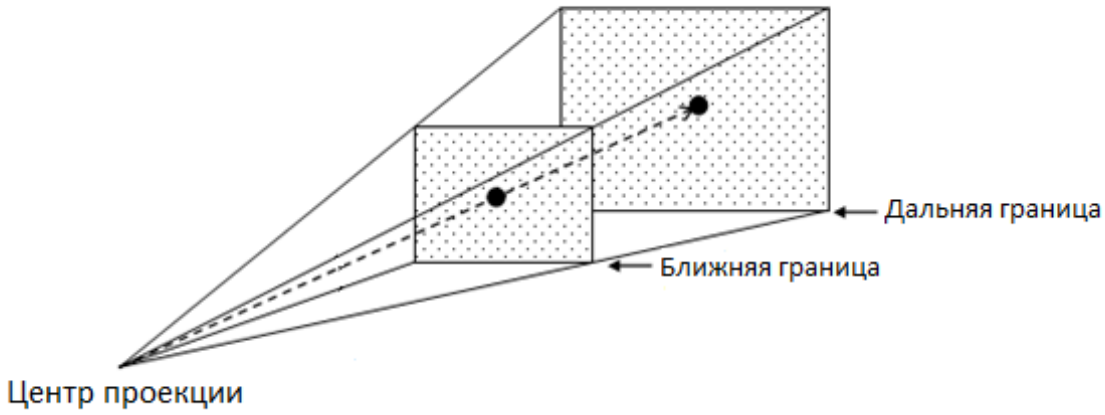


Рисунок 4.2 – Пирамида охвата видимости

После умножения видовых координат на матрицу проекции получаются усеченные координаты (clip coordinates). Как только координаты всех вершин будут переведены, выполняется заключительная операция, называемая перспективное деление. Разделив каждую координату (x, y, z) на однородную компоненту вектора w, получаются нормализованные координаты устройства (Normalize Device Coordinates) в диапазоне (-1.0, 1.0) по всем осям, поле вида превратилось в куб. Далее координаты сдвигом и масштабированием преобразуются в оконные координаты (window coordinates), которые уже участвуют в построении 2D-изображения на экране.

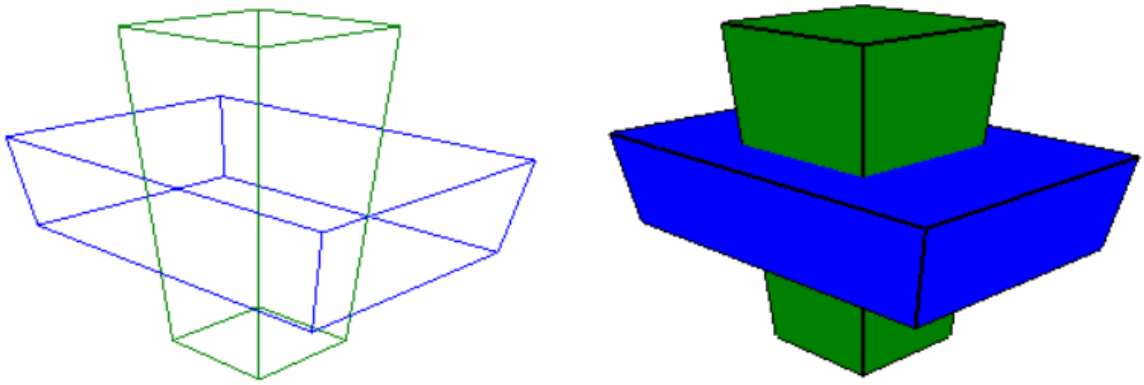
4.1. АЛГОРИТМЫ УДАЛЕНИЯ НЕВИДИМЫХ ЛИНИЙ

Реализация ДПК начинается с программирования алгоритмов. После этого необходимо отладить технологию и сформировать графический интерфейс по заданным требованиям.

В качестве объектов, для которых будут применяться алгоритмы, могут выступать только многогранники с гранями в виде выпуклых многоугольников, т.к. это ограничение диктуется особенностями алгоритма Робертса. Для демонстрации работы алгоритмов выбран параллелепипед. Треугольная призма исключается по причине излишней простоты. Фигура с большим количеством граней, чем параллелепипед, может привести к недостаточно четкой визуализации. Входными данными для всех алгоритмов буду координаты вершин объектов сцены.

В качестве примера (тестовой сцены) для всех алгоритмов будет использована сцена, состоящая из нескольких пересекающихся параллелепипедов (рисунок 4.3). Объекты в сцене задаются с помощью графического примитива Polygon, который представляет собой закрашенный многоугольник, ограниченный замкнутой ломаной линий. С помощью тестовой сцены будут проверяться на работоспособность все разработанные алгоритмы. Исходный код инициализации объектов в тестовой сцене приведен в листинге А.1 приложения А.

На рисунке 4.4 изображена тестовая сцена в прямоугольной системе координат. Средняя точка всей сцены расположена в точке отсчета – начале координат. В дальнейших расчетах расположение объектов для всех сцен будет именно таким. Точка наблюдения, там, где это не указано дополнительно, будет иметь координаты (40; 40; 40) (рисунок 4.5)).



Каркасное изображение Желаемый результат

Рисунок 4.3 – Тестовая сцена

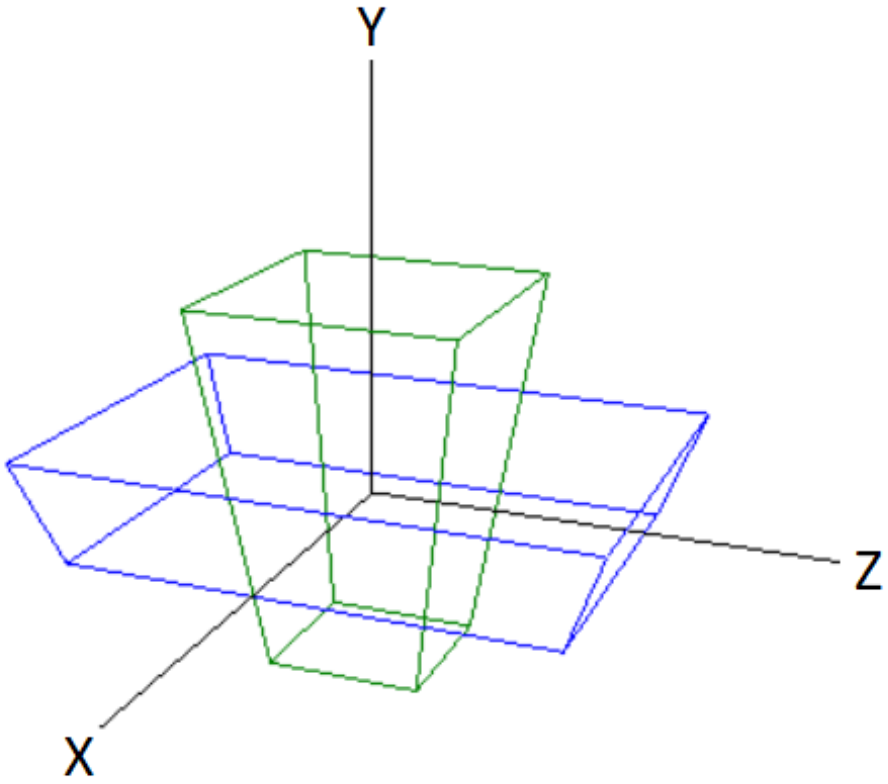


Рисунок 4.4 – Сцена в координатной системе

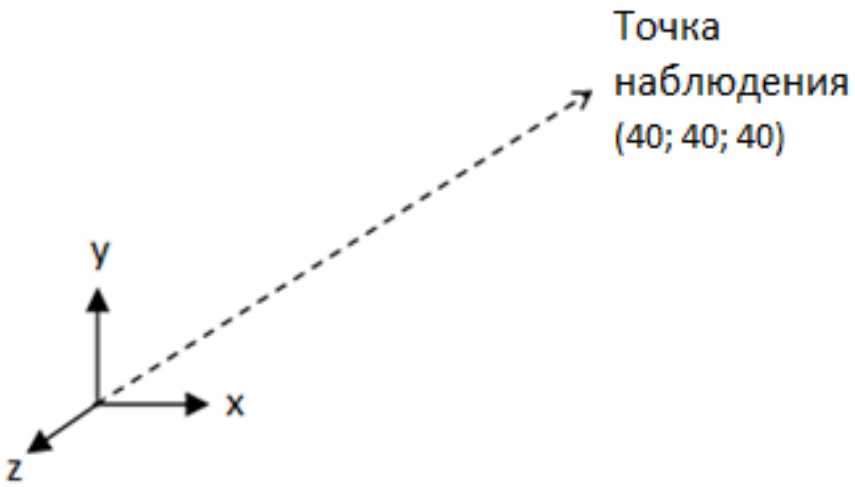


Рисунок 4.5 – Расположение точки наблюдения

4.1.1 Тест видимости

Выполнение каждого из алгоритмов начинается с теста видимости [1, 4]. Для каждого тела, вне зависимости от алгоритма, удаляются ребра, смыкающие две невидимые грани. Для применения метода требуется, чтобы все изображаемые тела или объекты были выпуклыми. Невыпуклые тела должны быть разбиты на выпуклые части.

Пусть F — некоторая грань многогранника. Тогда плоскость, в которой находится эта грань, разделяет пространство на два подпространства. Положительным будет считаться то из них, в которое смотрит внешняя нормаль к грани. Тогда если точка наблюдения находится в положительном подпространстве, то грань – лицевая, в противном случае – нелицевая. Для выпуклого многогранника удаление всех нелицевых граней полностью решает задачу удаления невидимых граней.

Для того, чтобы определить, лежит ли точка в положительном подпространстве, проверяется знак скалярного произведения: $(\bar{l},\ \bar{n})$, где $\bar{l}$ –

вектор, направленный к наблюдателю, фактически определяет точку наблюдения; $\bar{n}$ – вектор внешней нормали грани.

В случае $(\bar{l},\ \bar{n}) > 0$ угол между векторами острый и грань является лицевой. Если $(\bar{l},\ \bar{n}) < 0$, то угол между векторами тупой, а грань – нелицевая.

Уравнение произвольной плоскости в трехмерном пространстве имеет вид: $ax + by + cz + d = 0$. В матричной форме этот результат выглядит так:

$$[x\ y\ z\ w] * [P]^T = 0, \text{ где } [P]^T = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}.$$

Тогда любое выпуклое твердое тело может быть выражено матрицей тела, состоящей из коэффициентов уравнений плоскостей:

$[V] = \begin{bmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{bmatrix}$, где каждый столбец содержит коэффициенты одной плоскости.

Любая точка пространства представима в однородных координатах вектором $\bar{s} = (x, y, z, 1)$. Если точка лежит на плоскости, то $[\bar{s}] * [P]^T = 0$, в противном случае знак скалярного произведения показывает, по какую из сторон от плоскости расположена точка. Далее в расчетах предполагается, что точки, которые лежат внутри тела, дают отрицательное скалярное произведение, т. е. нормали к граням направлены наружу.

В качестве примера рассматривается единичный куб с центром в начале координат (рисунок 4.6).

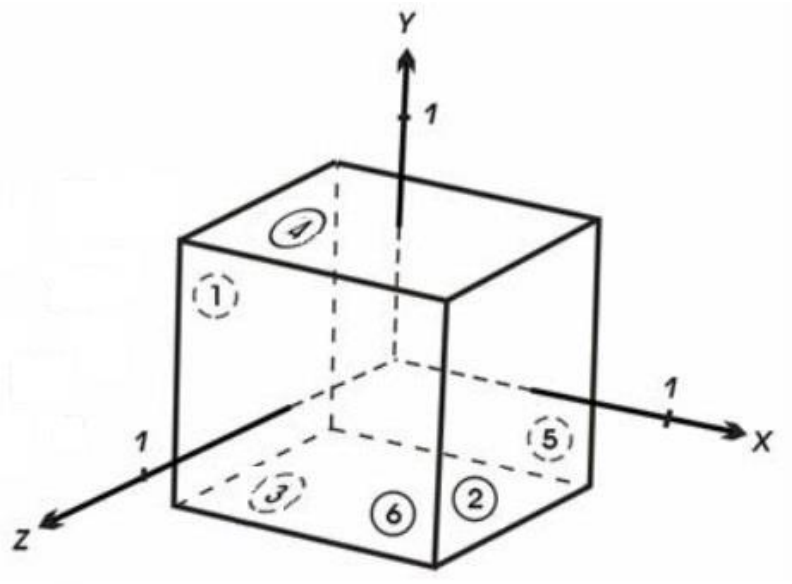


Рисунок 4.6 – Куб с центром в начале координат

Шесть плоскостей, описывающих данный куб:

$$x_1 = -\frac{1}{2};\ x_2 = \frac{1}{2};\ y_3 = -\frac{1}{2};\ y_4 = \frac{1}{2};\ z_5 = -\frac{1}{2};\ z_6 = \frac{1}{2}.$$

Полная матрица тела такова:

$$[V] = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} = \begin{bmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}.$$

Далее проверяется правильность матрицы тела с помощью точки, лежащей внутри тела. Если знак скалярного произведения этой точки и матрицы тела для какой-нибудь из плоскостей больше нуля, то соответствующее уравнение плоскости следует умножить на -1.

Например, точка, лежащая внутри куба, имеет координаты:

$x = \frac{1}{4}, y = \frac{1}{4}, z = \frac{1}{4}.$

В однородных координатах эта точка представляется в виде вектора:

$$[\bar{s}] = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 1 \end{bmatrix} = [1 \quad 1 \quad 1 \quad 4].$$

Скалярное произведение этого вектора на матрицу тела равно:

$$[\bar{s}] * [V] = [1 \quad 1 \quad 1 \quad 4] * \begin{bmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix} =$$

$[6 \quad -2 \quad 6 \quad -2 \quad 6 \quad -2]$.

Здесь результаты для 1-го, 3-го и 5-го уравнения плоскостей (столбцов) положительны и, следовательно, составлены некорректно. После умножения этих столбцов на -1, получается корректная матрицу тела для единичного куба:

$$[V] = \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}.$$

Плоскости имеют бесконечную протяженность и скалярное произведение радиус-вектора точки на матрицу тела положительно в том случае, если точка лежит вне этого тела. Таким образом матрица тела может использоваться для определения граней, которые экранируются самим телом (рисунок 4.7).

Условие $E_i * V_i < 0$ определяет, что плоскости – нелицевые, а их грани – задние.

$$[E] * [V] = [0 \quad 0 \quad 1 \quad 0] * \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

В результате произведения $[E] * [V]$ получается следующая однострочная матрица: $[0 \quad 0 \quad 0 \quad 0 \quad -2 \quad 2]$.

Положительный результат вектора E и вектора нормали можно интерпретировать как острый угол между этими векторами, отрицательный результат – как тупой угол. Если угол между векторами острый, то грань является лицевой, если угол тупой, то грань – нелицевая.

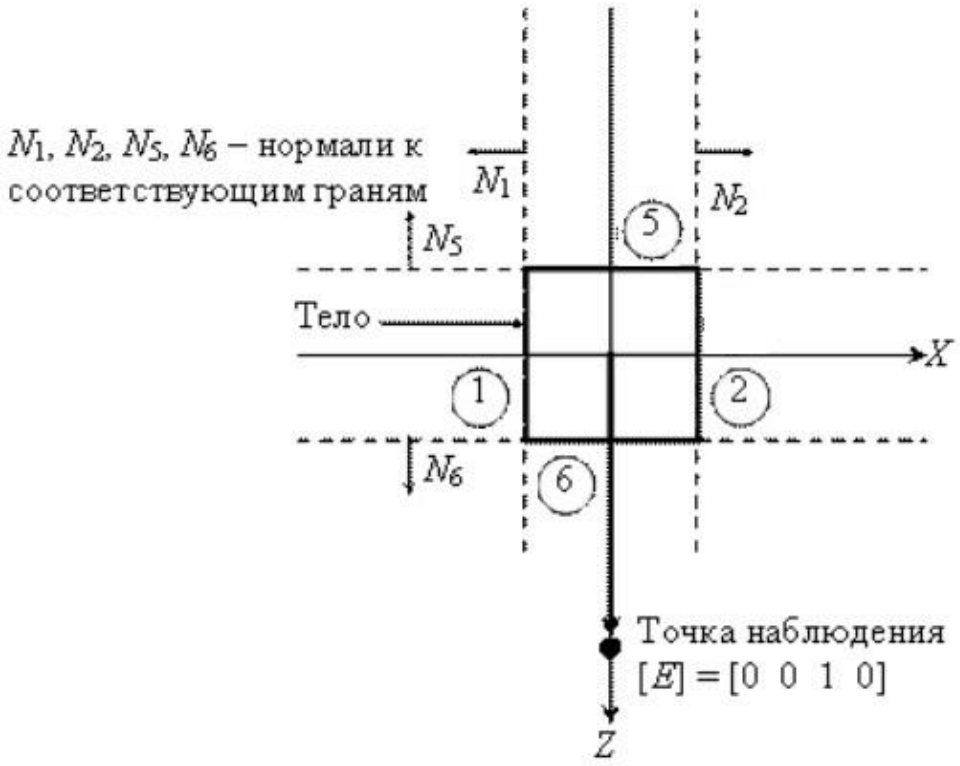


Рисунок 4.7 – Точка наблюдения вне тела

Положительное число в шестом столбце показывает, что грань лицевая. Отрицательное число в пятом столбце показывает, что грань с этим номером нелицевая. Нулевые результаты соответствуют плоскостям, параллельным направлению взгляда.

Рассмотрим выполнение теста видимости для одной из фигур в тестовой сцене (рисунок 4.8).

$$[V] = \begin{bmatrix} -363 & 363 & 0 & 0 & 0 & 0 \\ 0 & 0 & -674 & 674 & 0 & 0 \\ 0 & 0 & 0 & 0 & -524 & 524 \\ -5661 & -5661 & -5661 & -5661 & -5661 & -5661 \end{bmatrix}$$ –

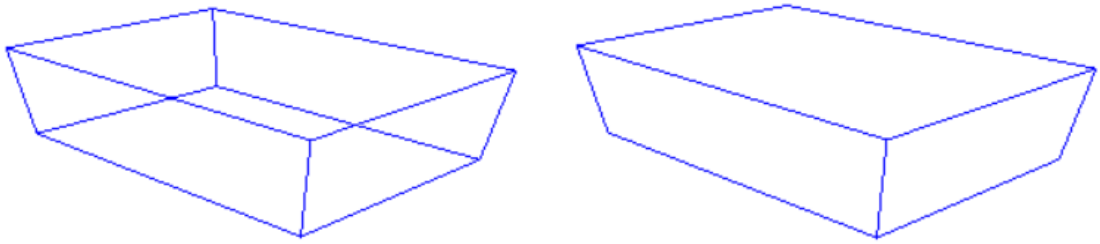
матрица тела объекта.

$[E] = [40 \quad 40 \quad 40 \quad 1]$ – точка наблюдения.

Их произведение дает следующую однострочную матрицу:

$$[-20176 \quad 8854 \quad -32617 \quad 21295 \quad -26627 \quad 15305].$$

Положительные результаты в столбцах 2, 4, 6 показывают, что грани под этими номерами лицевые, отрицательные результат в столбцах 1, 3, 5 – грани под этими номерами нелицевые. Так как задано только одно тело, алгоритм завершается.



Объект до теста видимости Объект после теста видимости

Рисунок 4.8 – Тест видимости для тестовой сцены

Тест видимости является простейшим методом удаления невидимых поверхностей для тел, которые представляют собой одиночные выпуклые многогранники. Он часто используется для удаления нелицевых граней из сцены перед применением одного из алгоритмов удаления невидимых линий, которые обсуждаются ниже. Для выпуклых многогранников число граней при этом сокращается примерно наполовину.

Абстрактный класс Algorithm содержит основные поля и методы, необходимые для выполнения алгоритмов загораживания. Одним из этих методов является doVisibilityTest(), который реализует тест видимости. Исходный код этого метода приведен в листинге Б.1 приложения Б.

4.1.2 Алгоритм Робертса

Алгоритм Робертса представляет собой одно из первых известных решений задачи об удалении невидимых линий [17, 19]. Он работает в объектном пространстве и требует, чтобы грани были выпуклыми многоугольниками.

Данный алгоритм переборного типа и работает с ребрами, то есть больше подходит для каркасных изображений. В этом случае изображение формируется видимыми ребрами. Ребра, которые смыкают две невидимые грани, не рассматриваются, т.к. они заведомо невидимы.

Работа Алгоритм Робертса проходит в два этапа:

1. Определение невидимых граней для каждого тела в сцене отдельно (тест видимости).
2. Определение и удаление невидимых ребер.

После первого этапа удаления нелицевых граней (рисунок 4.9) определяется существование потенциально видимых ребер, которые экранируются другими телами в сцене.

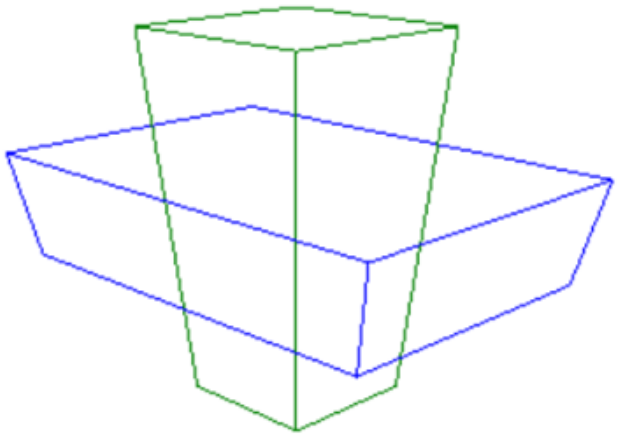


Рисунок 4.9 – Применение теста видимости для тестовой сцены

Каждое из оставшихся ребер в сцене проверяется на взаиморасположение с гранями каждого из остальных объектов. Целью считается определение частей ребра, которые экранируются этими объектами.

Возможны следующие случаи расположения:

1. *Грань ребра не закрывает*. Ребро остается в списке ребер.
2. *Грань полностью закрывает ребро*. Ребро удаляется из списка рассматриваемых ребер.
3. *Грань частично закрывает ребро*. В этом случае ребро разбивается на несколько частей, видимыми из которых являются не более двух. Само ребро удаляется из списка рассматриваемых ребер, но в список проверяемых ребер добавляются те его части, которые данной гранью не закрываются.

Обозначим исходные данные перед применением алгоритма:

P_1P_2 – исследуемое ребро.

$P(t) = P_1 + t * (P_2 - P_1)$ или $\bar{v} = \bar{s} + t * \bar{d}$ – параметрическое уравнение прямой, на которой лежит исследуемое ребро, где $\bar{s} = P_1$, $\bar{d} = (P_2 - P_1)$.

Необходимо найти такое t, при котором изменяется видимость ребра.

Параметрическое задание отрезка от точки $\vec{v}$ до $\vec{g}$ – точки наблюдения:

$Q(a, t) = \vec{v} + a * \bar{g},$ где $0 \leq a$.

Таким образом $Q(a, t) = \bar{s} + t\bar{d} + a\bar{g}$ – фактически является уравнением плоскости.

Скалярное произведение любой точки, расположенной внутри объекта и матрицы объекта, положительно (это утверждение справедливо и для преобразованной матрицы объекта). Точка, находящаяся внутри объекта, невидима. Следовательно, для проверки на экранирование вектор текущей точки отрезка умножают поочередно на матрицу каждого объекта и определяют

положительное решение, соответствующее прохождению отрезка внутри объекта:

Найдем такие t и a, для которых:

$\bar{H} = \left(\bar{s} + t * \bar{d} + a * \bar{g}\right) * [V] > 0$

или $\bar{H} = \bar{s} * [V] + t * \bar{d} * [V] + a * \bar{g} * [V] > 0.$

Если все компоненты $\bar{H}$ для некоторых t и a неотрицательны, то отрезок при этих t экранируется телом.

Обозначим:

$\bar{p} = \bar{s} * [V];\ \bar{q} = \bar{d} * [V];\ \bar{w} = \bar{g} * [V];$

Тогда:

$h_i = p_i + t * q_i + a * w_i > 0,$ где $0 \le t \le 1, 0 \le a, i -$ номер грани.

Далее нужно решить задачу линейного программирования при n неизвестных и m ограничителях.

После приравнивания h_i к 0 необходимо получить всевозможные пары систем 2-х уравнений, при n гранях число решений $\frac{n(n-1)}{2}$. Учитывая ограничения $0 \le a, t \le 1, 0 \le t$ будет $\frac{(n+3)(n-2)}{2}$ всевозможных решений.

Сначала находятся t и a для одной пары уравнений и подставляются в остальные.

Затем рассчитываются t_{min} – максимальное среди минимальных значений t (которые ближе к 0) и t_{max} – минимально среди максимальных значений t (которые ближе к 1). Отрезок невидим при $t_{min} < t < t_{max}$.

До того, как решать систему можно заранее определить факт полной видимости отрезка:

– $p_i < 0$ – 1-ая першина отрезка лежит по внешнюю сторону от i-ой грани;

– $p_i + q_i < 0$ – 2-ая вершина отрезка лежит по внешнюю сторону от i-ой грани;
– $w_i \leq 0$ – i-ая грань лицевая (видима);
– $w_i \leq 0$ и $p_i \leq 0$ – один конец отрезка лежит или на видимой плоскости или между видимой плоскостью и точкой наблюдения;
– при $w_i \leq 0$, $p_i \leq 0$ и $p_i + q_i \leq 0$ гарантируется, что отрезок полностью видим.

Применение алгоритма можно разделить на следующие этапы:

1. Определение коэффициентов уравнения плоскости каждой грани, проверка правильности знака уравнения и формирование матрицы объекта визуализации.
2. Определение нелицевых граней, удаление их из списка граней и соответствующих ребер - из списка ребер.
3. Определение невидимых отрезков или участков отрезков.
4. Формирование списка возможных отрезков, соединяющих точки протыкания, для пар объектов, связанных отношением протыкания.
5. Проверка видимости полученных отрезков по отношению ко всем объектам сцены в соответствии с этапами 2 и 3.
6. Визуализация изображения.

Блок-схема алгоритма Робертса представлена на рисунке 4.10.

В качестве примера применения алгоритма рассматривается одно из ребер в тестовой сцене (рисунок 4.11).

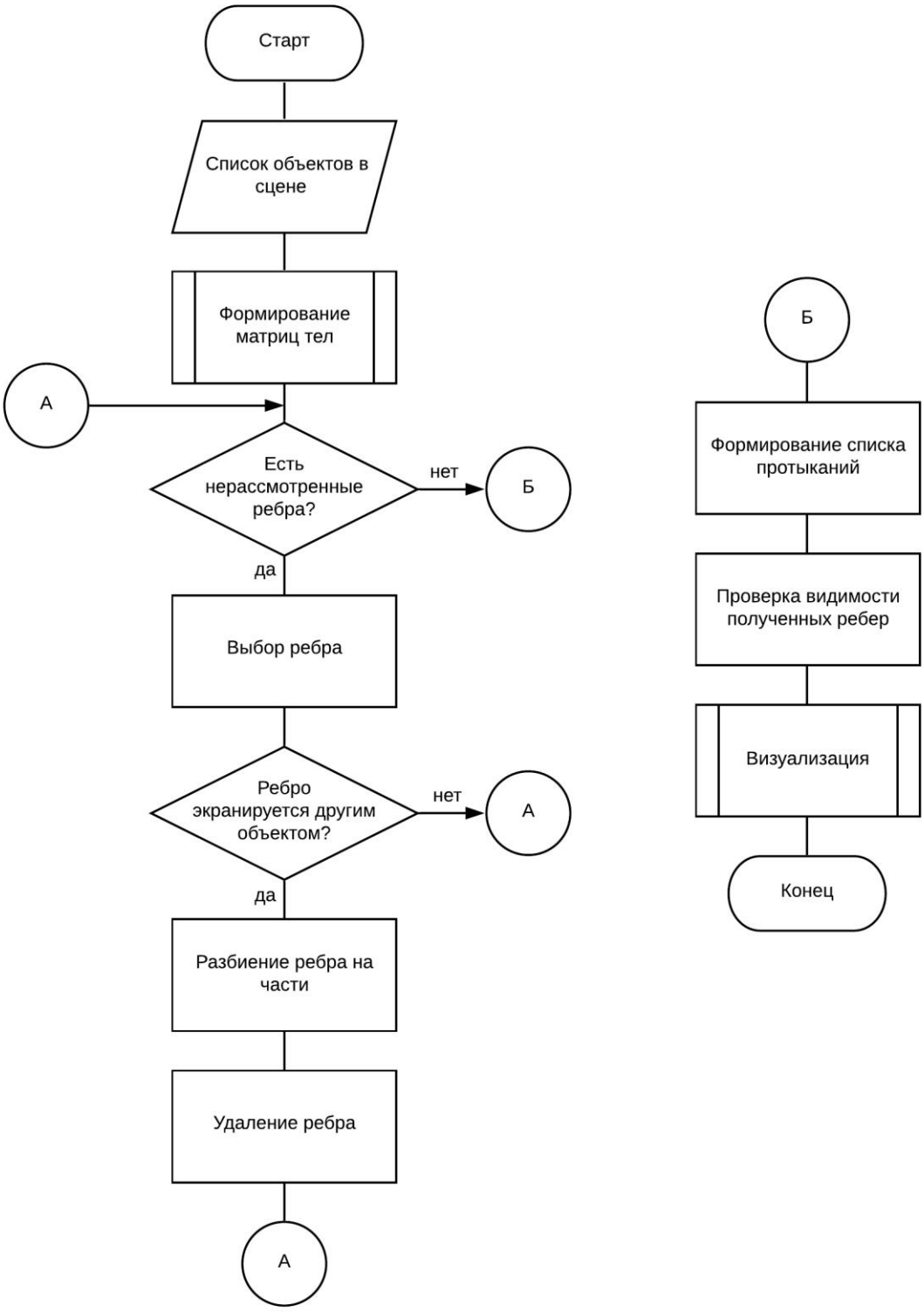


Рисунок 4.10 – Блок-схема алгоритма Робертса

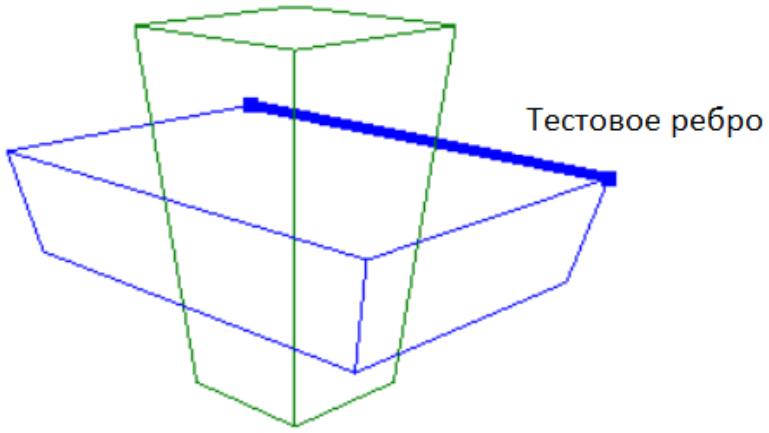


Рисунок 4.11 – Алгоритм Робертса для выбранного ребра

Исходные данные:

$P_н = (-15{,}6;\ 8{,}4;\ -10{,}8)$, $P_к = (15{,}6;\ 8{,}4;\ -10{,}8)$ – тестируемое ребро.

$$[V] = \begin{bmatrix} 720 & -720 & 0 & 0 & 0 & 0 \\ 0 & 0 & 720 & -720 & 0 & 0 \\ 0 & 0 & 0 & 0 & 720 & -720 \\ 4320 & 4320 & 5011 & 3628 & 4320 & 4320 \end{bmatrix}$$ – матрица тела зеленого объекта (нормали к граням направлены внутрь тела);

$\bar{d} = (31.2;\ 0;\ 0;\ 0);\ \bar{s} = (-15{,}6;\ 8{,}4;\ -10{,}8);\ \bar{g} = (40;\ 40;\ 40);$

$\bar{p} = \bar{s} * [V] = (-6912;\ 15552;\ 6220;\ 2419; -3456;\ 12096);$

$\bar{q} = \bar{d} * [V] = (22464; -22464;\ 0;\ 0;\ 0;\ 0);$

$\bar{w} = \bar{g} * [V] = (33120;\ -24480; 10771;\ -2131; 33120;\ -24480);$

Система неравенств $h_i = p_i + t * q_i + a * w_i > 0$ выглядит следующим образом:

$$\begin{cases} -6912 + t * 22464 + a * 33120 > 0 \\ 15552 + t * -22464 + a * -24480 > 0 \\ 6220{,}8 + t * 0 + a * 10771{,}2 > 0 \\ 2419{,}2 + t * 0 + a * -2131{,}2 > 0 \\ -3456 + t * 0 + a * 33120 > 0 \\ 12096 + t * 0 + a * -24480 > 0 \end{cases}$$

При условии, что $h_i = 0$ каждое из неравенств фактически становится уравнением плоскости. Пара уравнений задают прямую, принадлежащую обеим плоскостям. Необходимо получить всевозможные пары систем 2-х уравнений. Число решений в таком случае равно $\frac{n(n-1)}{2}$ или $\frac{6*5}{2} = 15$.

Учитывая ограничения $0 \leq a, t \leq 1, 0 \leq t$ были найдены несколько пар:

1) $t = 0.154$ и $a = 0{,}494$;

2) $t = 0.579$ и $a = 0.104$;

По найденным данным рассчитываются вершины:

$$P_1 = P(0.154) = (-15{,}6\ \ 8{,}4\ \ -10{,}8) + (31.2\ 0\ 0\ 0) * 0.154 = (-10.8;\ \ 8{,}4;\ -10{,}8);$$

$$P_2 = P(0.579) = (-15{,}6\ \ 8{,}4\ \ -10{,}8) + (31.2\ 0\ 0\ 0) * 0.579 = (2{,}45;\ 8{,}4;\ -10{,}8);$$

Найденные вершины образуют новые отрезки с начальной и конечной вершинами тестового ребра. При этом единственным видимым отрезком является $P_2 P_к$ – новое видимое ребро.

На рисунке 4.12 толстой линией выделена видимая часть ребра, а также грань, относительно которой изменяется видимость ребра.

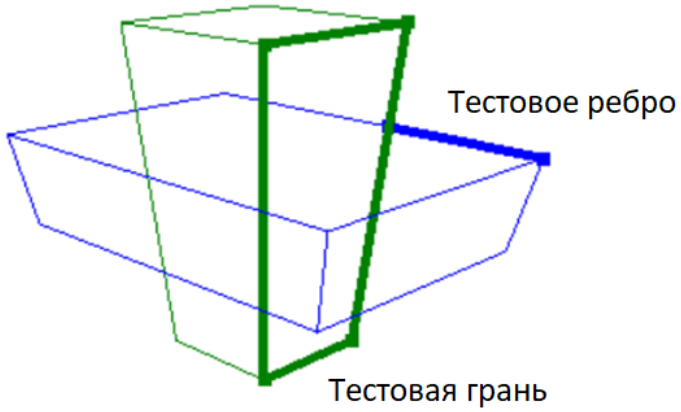


Рисунок 4.12 – Применение алгоритма Робертса для тестового ребра

После определения частично видимых или полностью невидимых отрезков (рисунок 4.13) определяют пары объектов, связанных отношением протыкания.

Для этого необходимо формировать уравнения $h_i = 0$ и решить их, объединяя попарно и включив в систему уравнения границ $t = 0$ и $t = 0$, а также включив в систему уравнение границы $a = 0$. Точки протыкания запоминаются.

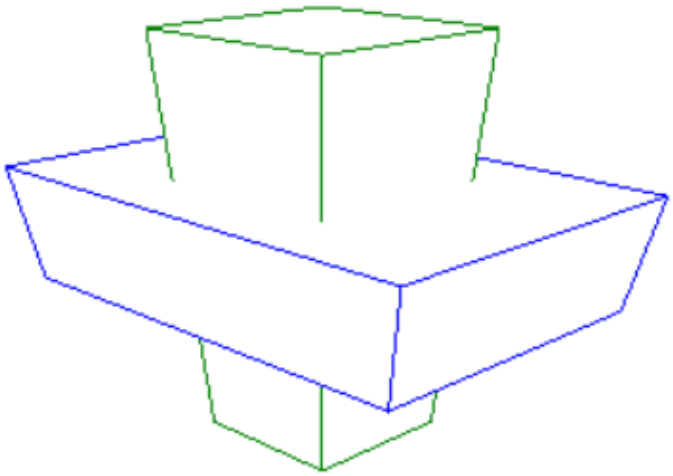
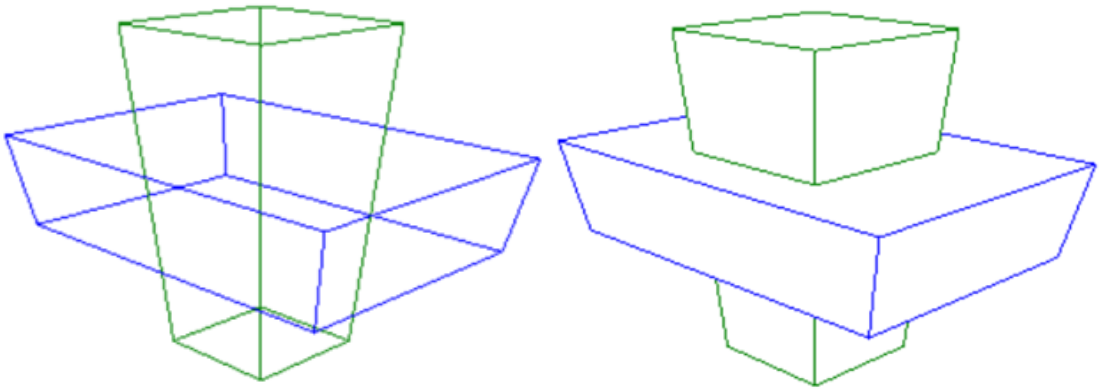


Рисунок 4.13 – Невидимые ребра удалены

Если точек протыкания не обнаружено, то происходит переход к процедуре визуализации. В противном случае необходимо определить видимые отрезки, связывающие точки протыкания и затем сформировать все возможные ребра, соединяющие точки протыкания, для пар тел, связанных отношением протыкания. Далее проверяется экранирование всех соединяющих ребер обоими телами, связанными отношением протыкания. В конечном счете выполняется переход к процедуре визуализации.

На рисунке 4.14 представлена иллюстрация применения алгоритма Робертса ко всей тестовой сцене.

В ходе тестирования работоспособности алгоритма обрабатывалась отладочная информация, выводимая в консоль (пример, приложение В).



Каркасное изображение Изображение на выходе

Рисунок 4.14 – Применение алгоритма Робертса

Временные затраты алгоритма пропорциональны квадрату числа граней всех объектов $O(n^2)$. Данный факт привёл к снижению интереса к алгоритму Робертса. Однако математические методы, используемые в этом алгоритме, просты, мощны и точны. Кроме того, этот алгоритм можно использовать для иллюстрации некоторых важных концепций.

Исходный код основных методов класса Roberts представлен в приложении Г. Функционал основных методов следующий:

– RemoveHiddenEdges() (листинг Г.1) организует проход по всем граням объектов в сцене. Также здесь анализируется список протыканий для пар объектов, связанных отношением протыкания ;
– CheckVisibilityOfEdges() (листинг Г.2) реализует логику проверки видимости ребер. В качестве одного из аргументов принимается список всех ребер очередной грани. В процессе применения алгоритма список может пополняться новыми созданными ребрами, все невидимые ребра – удаляются.

– incrementPhase(), decrementPhase(), drawPhase() (листинги Г.3-Г.5) – переопределение методов абстрактного класса, отвечающих за перемещение между этапами алгоритма, их отрисовку.

4.1.3 Алгоритм Аппеля

Метод Аппеля [18, 19] применяется для полигональных моделей. Он основан на подсчете количественной невидимости – числа лицевых граней, закрывающих данную точку поверхности. Точка является видимой в том случае, если ее количественная невидимость равна нулю.

Количественная невидимость точки ребра изменяется при прохождении ребра позади так называемой контурной линии. Контурная линия состоит из ребер, которые смыкают две грани, одна из которых видимая (лицевая), а другая невидимая (нелицевая). На рисунке 4.15 контурная линия для каждого из объектов в сцене выделена своим цветом. Так как рассматривать стоит только потенциально видимые ребра, перед работой алгоритма был выполнен тест видимости.

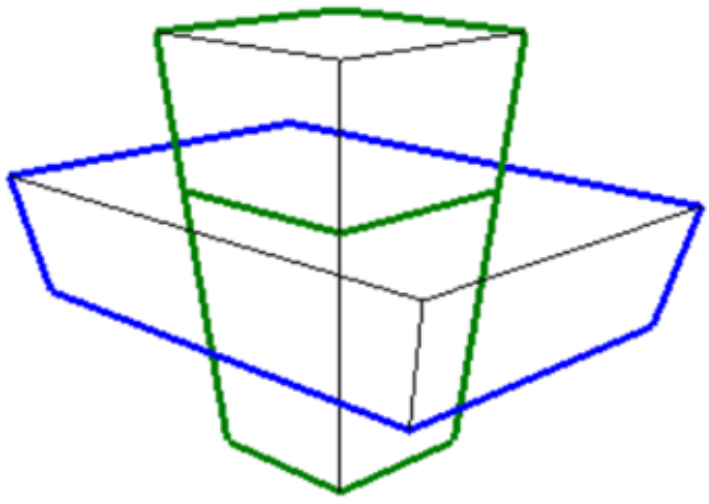


Рисунок 4.15 – Контурная линия для тестовой сцены

Количественная невидимость ребра увеличивается на 1 в том случае, если ребро уходит за контурное ребро и уменьшается на 1, если ребро выходит из-за контурного ребра. Таким образом ребро разбивается на отрезки с различными значениями количественной невидимости. Различие между соседними отрезками равно 1.

Применение алгоритма начинается с выбора какой-либо вершины многогранника. Для того чтобы определить ее количественную невидимость строится отрезок прямой через эту точку и точку наблюдения, находится пересечения этого отрезка с каждой гранью объекта. Число найденных граней – количественная невидимость начальной точки.

Следующим шагом определяется изменение количественной невидимости для всех ребер, которые выходят из новой вершины. Эти ребра также проверяются на прохождение позади контурной линии. Затем выбирается вершина, образуемая одним из рассмотренных ребер, процесс повторяется. Те части отрезка, для которых количественная невидимость равна нулю, сразу рисуются.

Для того чтобы определить пересечение рассматриваемого ребра с контурным Аппель предложил следующее (рисунок 4.16). Образуется треугольник, вершинами которого являются точка наблюдения и концы исследуемого ребра. Количественная невидимость ребра q изменяется, если точка пересечения контурного ребра с плоскостью треугольника лежит внутри треугольника, т.е. ребро протыкает треугольник. При положительном знаке векторного произведения контурного и рассматриваемого ребер количественная невидимость q увеличивается на 1, в противном случае уменьшается на 1

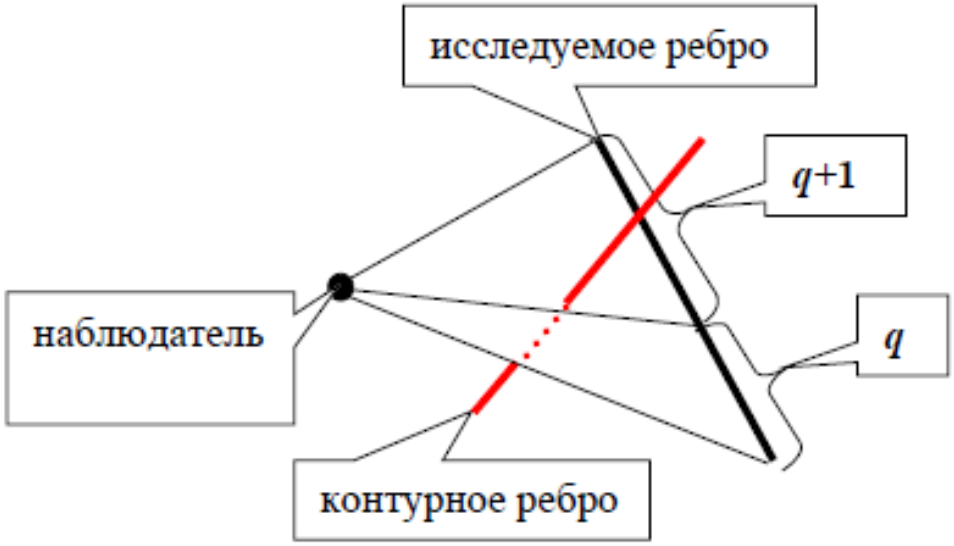


Рисунок 4.16 – Пересечение тестируемого ребра с контурным

Формально, применение алгоритма Аппеля можно разделить на следующие этапы:

1. Взять какую-либо вершину (не обязательно на контурной линии) и непосредственно определить ее количественную невидимость.
2. Проследить изменение количественной невидимости вдоль каждого из ребер, выходящих из этой вершины.
3. Проверить ребра на прохождение позади контурной линии: если ребро проходит позади контурной линии, то количественная невидимость точек ребра изменяется на единицу. Части отрезка, количественная невидимость которых равна нулю, сразу же рисуются.
4. Перейти к следующей вершине и определить количественную невидимость для ребер, выходящих из новой вершины.

Блох-схема алгоритма представлена на рисунке 4.17.

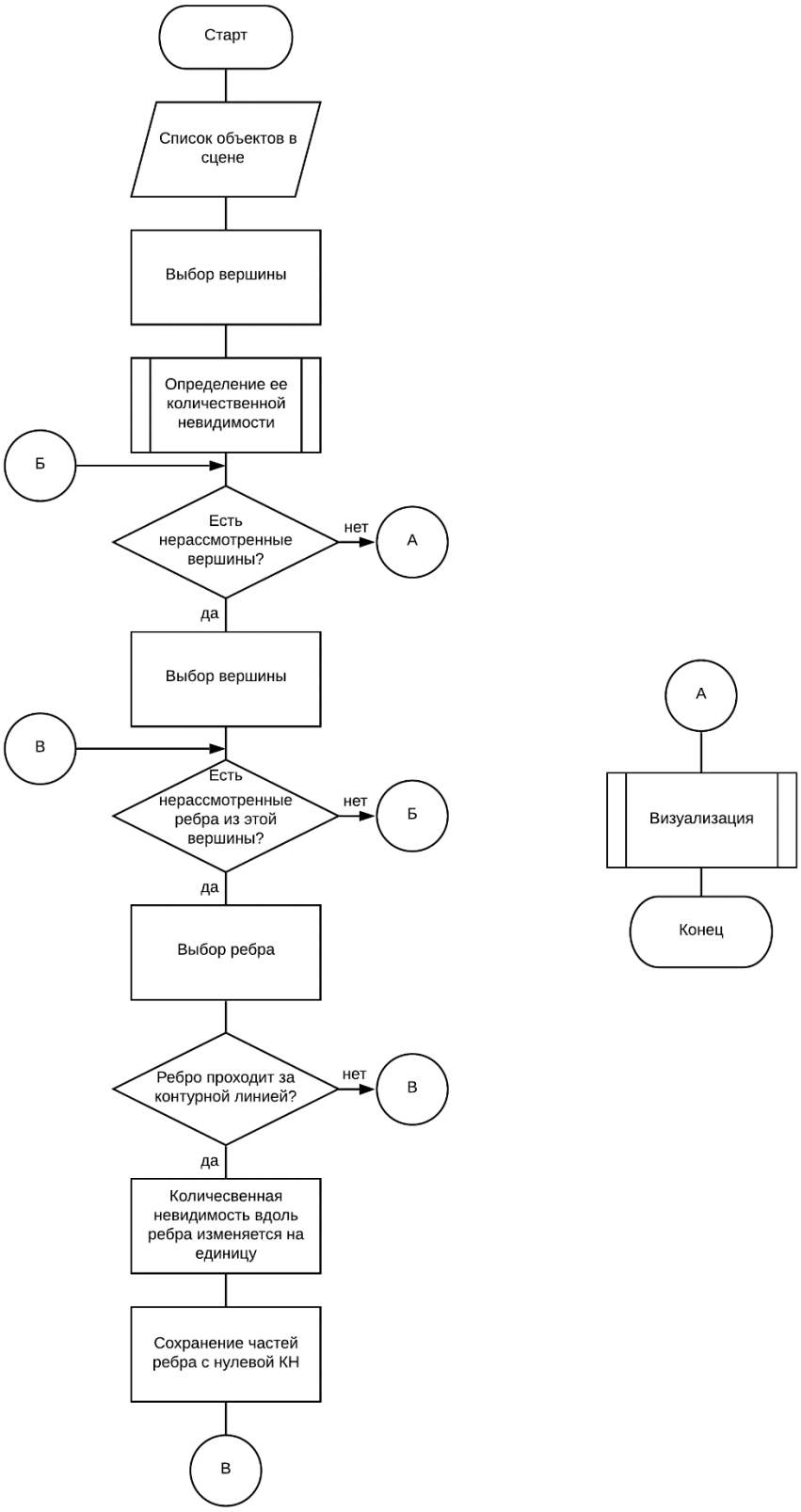


Рисунок 4.17 – Блок-схема алгоритма Аппеля

Далее рассматривается одно из ребер в тестовой сцене на предмет изменения видимости (рисунок 4.18).

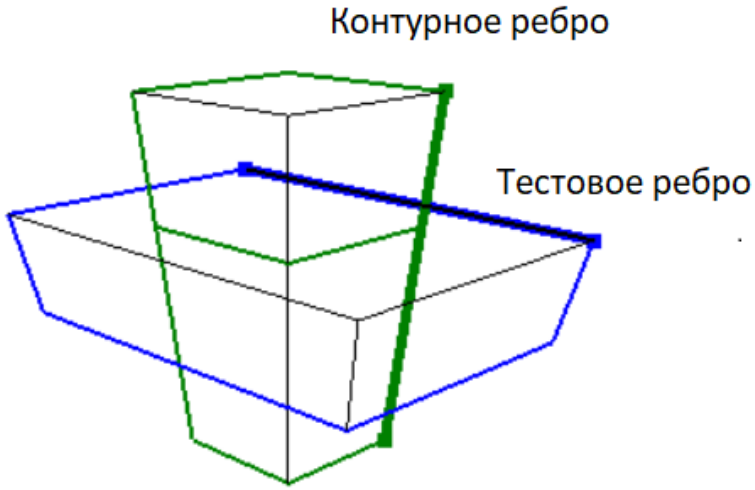


Рисунок 4.18 – Пересечение тестируемого ребра с контурным

После определения того, что тестовое ребро прошло за контурной линией, необходимо вычислить, какое влияние это оказало на его количественную невидимость. Как показано на рисунке 4.19, данная процедура выглядит следующим образом:

1. Проводятся линии визирования до вершин тестируемого ребра. Формируется треугольник, с которым необходимо определить точку пересечение контурного ребра. Так как треугольник строится из точки наблюдения, для того чтобы увидеть его необходимо зафиксировать положение точки наблюдения и затем изменить ее положение в пространстве.

2. Определяется точка прокалывания контурным ребром найденной плоскости. Данную точку необходимо спроецировать на тестовое ребро таким образом, что все три точки лежали на одной прямой.

3. Так как количественная невидимость начальной точки равна 1, когда тестовое ребро (P_1P_2) выходит из-за контурной линии количественная

невидимость должна быть уменьшена на единицу. Новый сегмент ребра имеет нулевую количественную невидимость, ребро отображается.

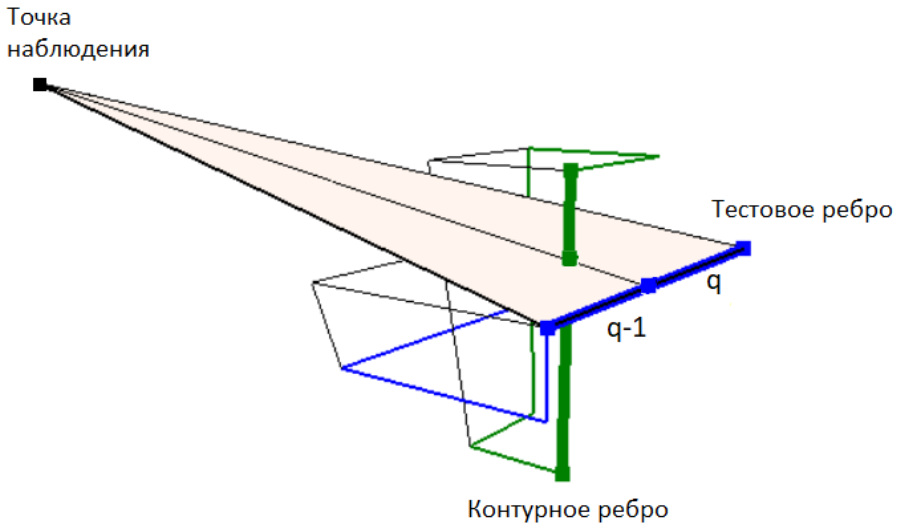


Рисунок 4.19 – Треугольник при алгоритме Аппеля

Результат применения алгоритма Аппеля ко всей сцене изображен на рисунке 4.20.

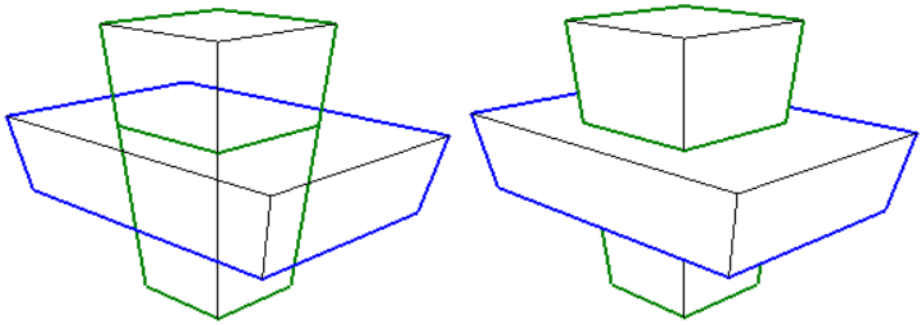


Рисунок 4.20 – Применение алгоритма Аппеля для тестовой сцены

По сравнению с алгоритмом Робертса алгоритм Аппеля более быстр (сложность $O(\sqrt{n})$, где n – общее число ребер), т.к. число ребер, входящих в контурную линию, намного меньше общего числа ребер.

Исходный код основных методов класса Appel приведен в приложении Д. Функционал основных методов следующий:

– RemoveHiddenEdges() (листинг Д.1) организует проход по всем ребрам объектов в сцене, для каждого строится свой треугольник из точки наблюдения. Каждое ребро, входящее в контурную линию, проверяется на предмет пересечения с треугольником. В случае пересечения находится изменение количественной невидимости тестового ребра;
– intersectionPointEdgeFace() (листинг Д.2) intersectionPointEdgeEdge() производят проверку пересечения ребра и грани, двух ребер. В качестве возвращаемого значения идет найденная точка пересечения;
– countQIForVertex() (листинг Д.3) определяет количественную невидимость точки как числа граней ее закрывающих;
– incrementPhase(), decrementPhase(), drawPhase() (листинги Д.4-Д.6) – переопределение методов абстрактного класса, отвечающих за перемещение между этапами алгоритма, их отрисовку.

4.1.4 Алгоритм Z-буфера

Алгоритм z-буфер [2,19] работает в пространстве изображения. Для его работы требуются два буфера: буфера глубины (Z-буфер) и буфера кадра. Буфер кадра используется для запоминания цвета каждого пикселя в пространстве изображения. Z-буфер или буфер глубины используется для запоминания координаты z или глубины каждого видимого пикселя в пространстве изображения.

В процессе работы алгоритма глубина или значение z каждого нового пикселя, который нужно занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в z-буфер. Если новое значение больше уже находящегося в буфере, то параметры нового пикселя заносятся в оба буфера.

Главное преимущество алгоритма – его простота. Алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей. Сцены при этом могут быть любой сложности. Элементы заносятся в буфер в произвольном порядке, их не нужно предварительно сортировать.

Основной недостаток алгоритма – большой объем требуемой памяти. Если сцена подвергается видовому преобразованию и отсекается до фиксированного диапазона значений координат z, то можно использовать z-буфер с фиксированной точностью.

Другой недостаток алгоритма z-буфера состоит в трудоемкости и высокой стоимости устранения лестничного эффекта, а также реализации эффектов прозрачности и просвечивания. Поскольку алгоритм заносит пиксели в буфер кадра в произвольном порядке, то нелегко получить информацию, необходимую для методов устранения лестничного эффекта, которые основываются на предварительной фильтрации.

Блок-схема алгоритма представлена на рисунке 4.21.

Формальное описание алгоритма z-буфера выглядит следующим образом:

1. Заполнить буфер кадра фоновым значением интенсивности или цвета.
2. Заполнить z-буфер минимальным значением *z*.
3. Преобразовать каждый многоугольник в растровую форму в произвольном порядке.
4. Для каждого пикселя в многоугольнике вычислить его глубину *z(x,y)*.
5. Сравнить глубину *z(x,y)* со значением *Z-буфер(x,y)*, хранящимся в z-буфере в этой же позиции. Если *z(x,y)* > *Z-буфер(x,y)*, то записать цвет этого многоугольника в буфер кадра и заменить *Z-буфер(x,y)* на *z(x,y)*. В противном случае перейти к следующему пикселю.

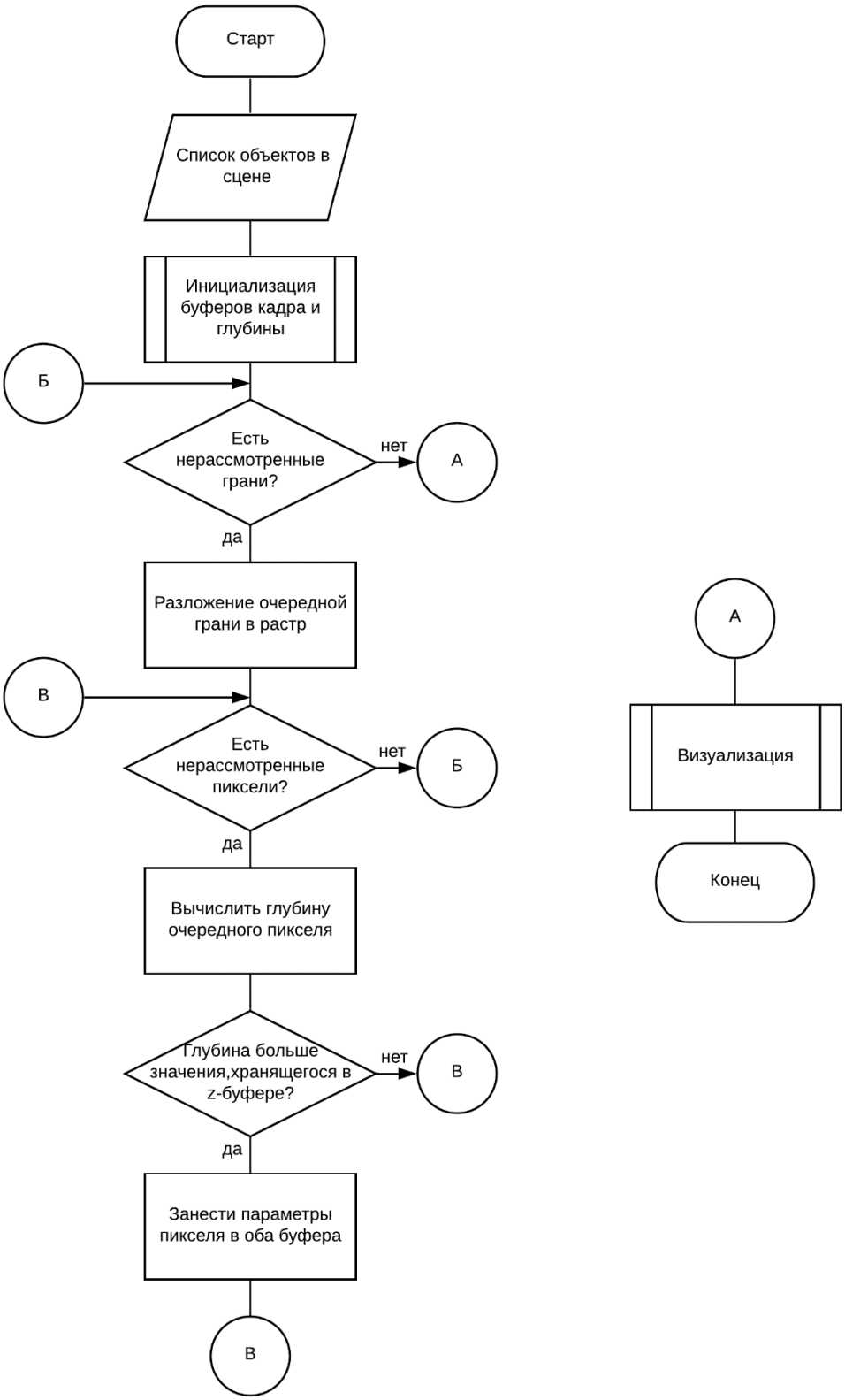


Рисунок 4.21 – Блок-схема алгоритма Z-буфера

В процессе работы алгоритма проецируемые области закрашиваются. Реализация алгоритма вдоль сканирующей строки позволяет совместить алгоритм z-буфера с алгоритмами закраски грани.

Если известно уравнение плоскости, несущей каждый многоугольник, то вычисление глубины каждого пикселя на сканирующей строке можно проделать пошаговым способом. Грань при этом рисуется строка за строкой. На рисунке 4.22 y меняется от y_1 до y_2, от y_2 до y_3, от y_3 до y_4, при этом для каждой строки определяется значения x_a, z_a, x_b, z_b:

$$x_a = x_1 - (x_2 - x_1) * \frac{y-y_1}{y_2-y_1}; \quad x_b = x_1 - (x_2 - x_1) * \frac{y-y_1}{y_3-y_1};$$

$$z_a = x_1 - (z_2 - z_1) * \frac{y-y_1}{y_2-y_1}; \quad z_b = z_1 - (z_3 - z_1) * \frac{y-y_1}{y_3-y_1};$$

На сканирующей строке x меняется от x_a до x_b и для каждой точки строки определяется глубина z:

$$z = z_a - (z_b - z_a) * \frac{x-x_a}{x_b-x_a};$$

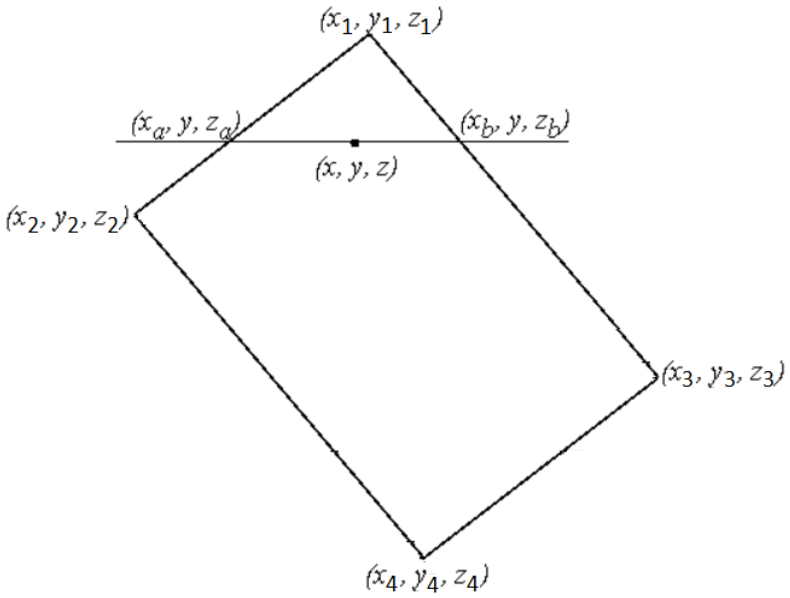


Рисунок 4.22 – Сканирующая строка

Для наглядной демонстрации работы алгоритма необходима область, куда будет проецироваться сцена. Проведем линию визирования, т.е. линию, проходящую через точку наблюдения и начало координат. Зададим плоскость прямоугольника по точке и вектору нормали. Точка в данном случае лежит на линии визирования, и отступает от начала координат на заданный отрезок, который и будет являться вектором нормали (рисунок 4.23).

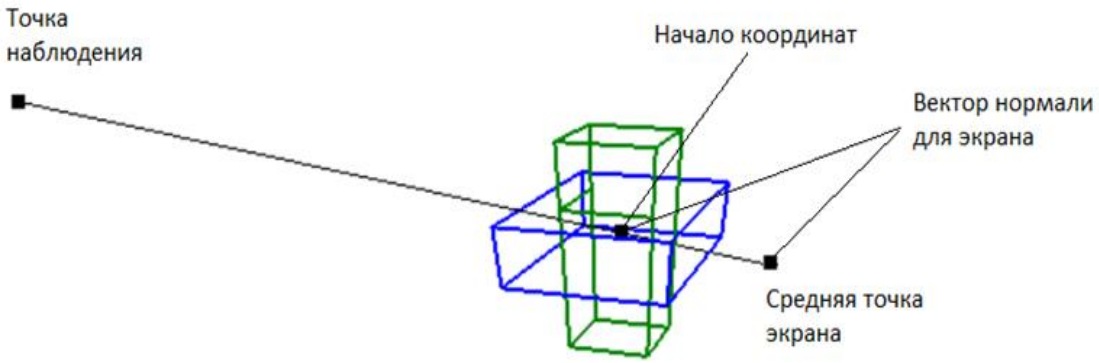


Рисунок 4.23 – Построение области для проецирования

Искомая область изображена на рисунках 4.24 и 4.25. Очевидно, что она перпендикулярна линии визирования. В данной работе эта область будет именоваться экраном.

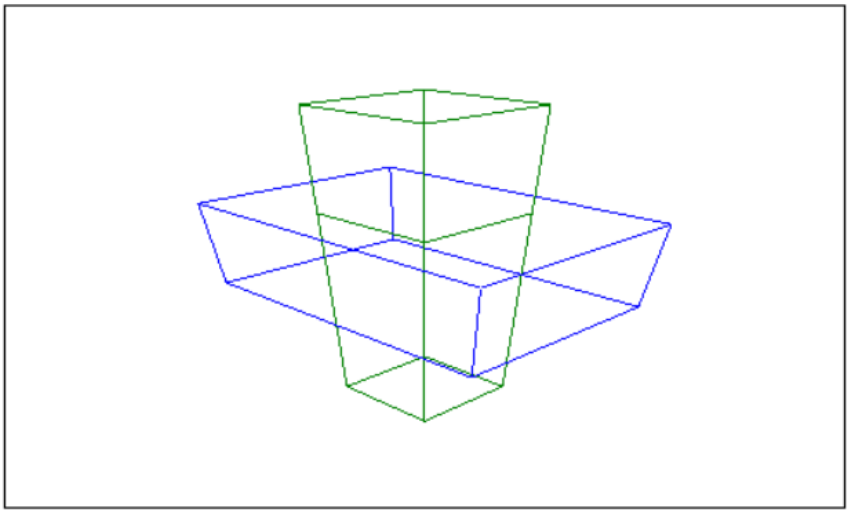


Рисунок 4.24 – Экран для проекций

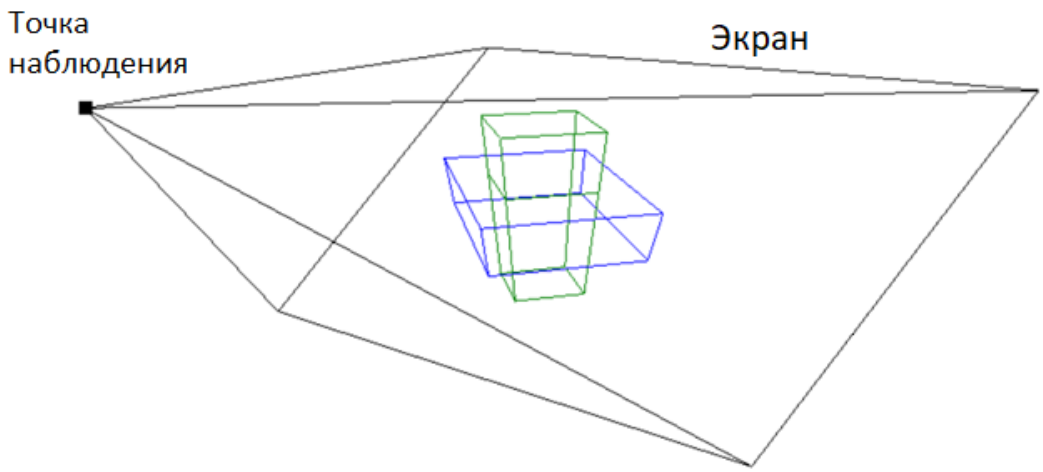


Рисунок 4.25 – Положение экрана относительно точки наблюдения

Каждая из граней объектов тестовой сцены проецируется на экран и разлагается там в растр. Используемый метод создания проекции – центральное проецирование, при котором все проецирующие лучи исходят из одной точки – центра проецирования. Разрешение экрана будет задаваться одним числом, при этом каждый пиксель будет иметь форму прямоугольника (рисунок 4.26). Растеризация всех ребер в проецируемой плоскости будет осуществляться с применением алгоритма Брезенхема [19].

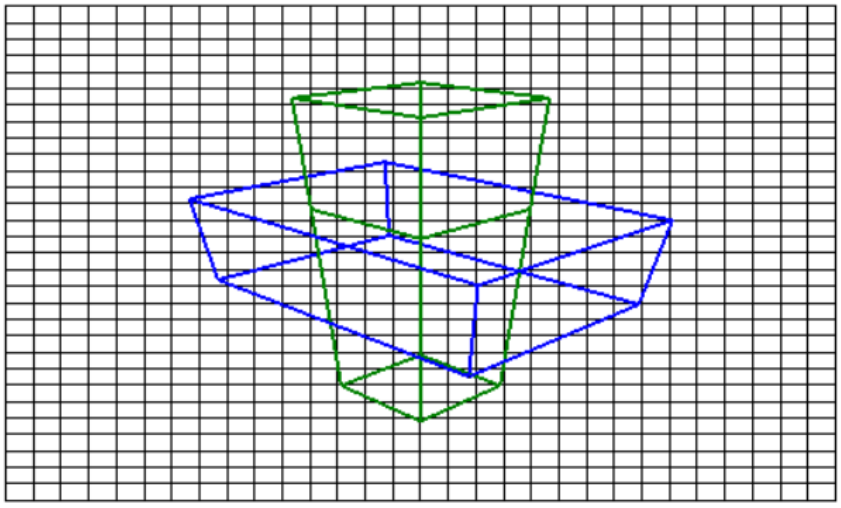


Рисунок 4.26 – Экран для проекции с заданным разрешением

Проиллюстрируем работу алгоритма на примере одной из граней в тестовой сцене. Пусть разрешение экрана будет составлять 30×30 пикселей (рисунки 4.27, 4.28). Изначально в буфере кадра и в z-буфере содержатся нулевые значения. После растровой развертки рассматриваемой грани в оба буфера вносятся новые значения. Конечное содержимое буферов отображено в приложении Е.

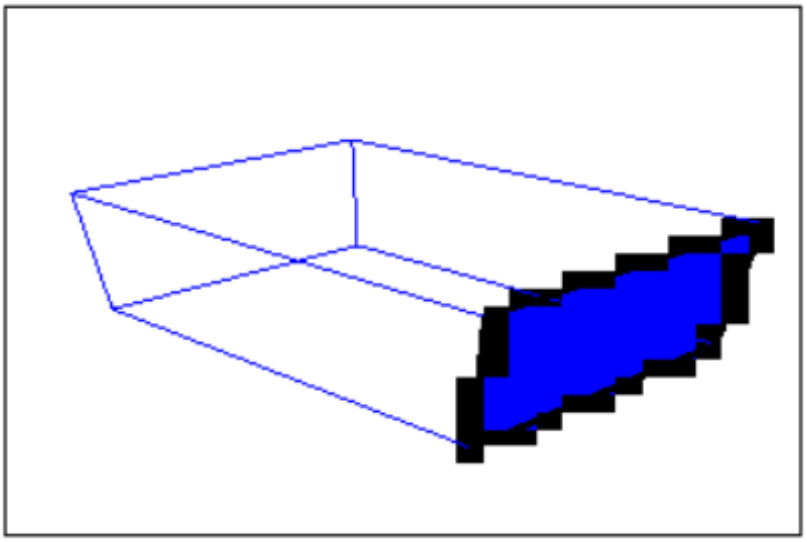


Рисунок 4.27 – Пример работы алгоритма

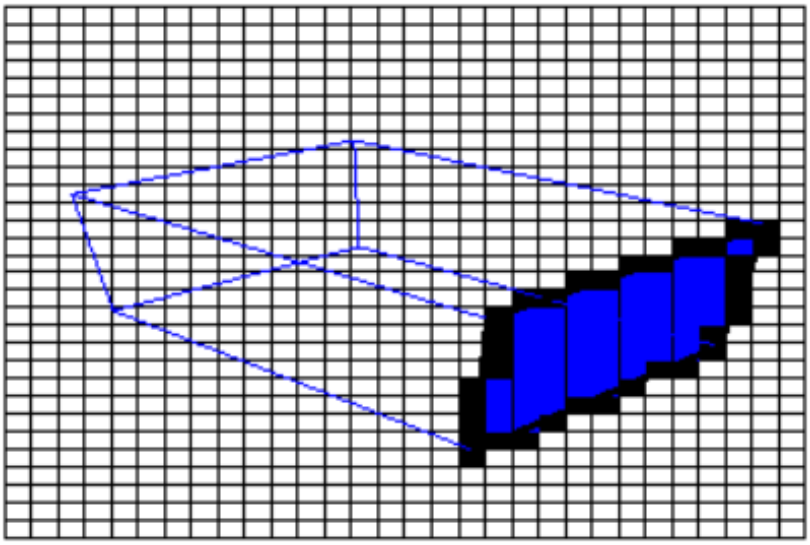


Рисунок 4.28 – Пример работы алгоритма

На рисунке 4.29 представлен процесс проецирования тестовой грани.

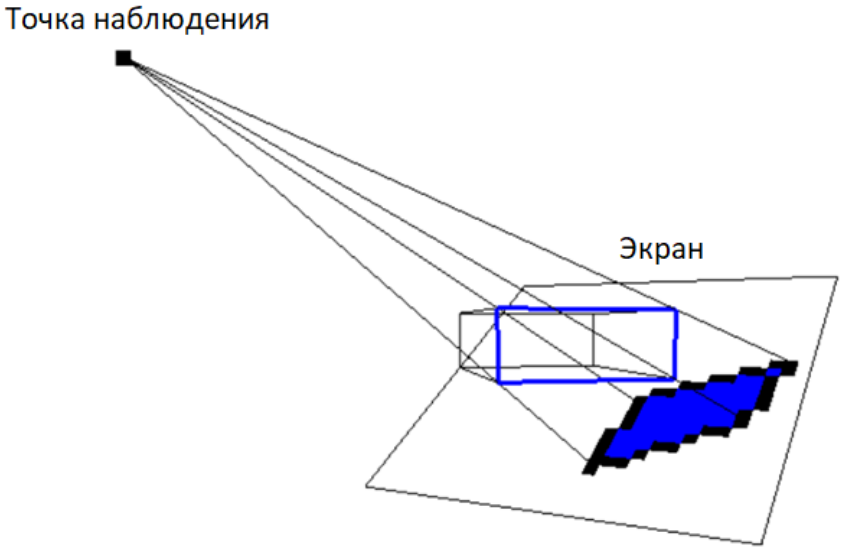


Рисунок 4.29 – Пример работы алгоритма

Для проверки работоспособности алгоритма необходимо применить его ко всей тестовой сцене. Результат этого применения изображен на рисунке 4.30. Разрешение экрана в данном случае равно 50×50. Значения, хранящиеся в буферах глубины и кадра по итогу работы алгоритма представлены в приложении Ж.

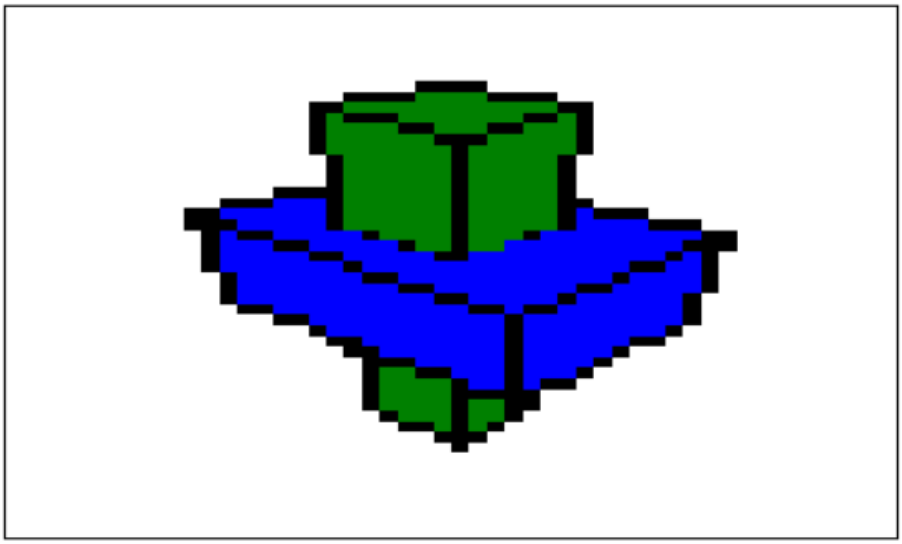


Рисунок 4.30 – Применение алгоритма для тестовой сцены

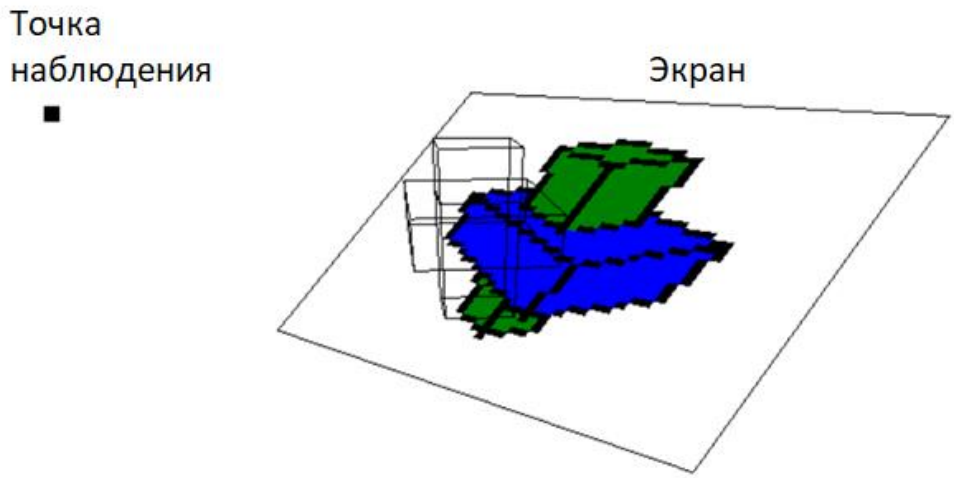


Рисунок 4.31 – Применение алгоритма для тестовой сцены

Для сравнения был приведен результат применения алгоритма для той же сцены, но уже с разрешением экрана 250×250 пикселей (рисунки 4.32, 4.33). Очевидно, что изображение стало четче в связи с отсутствием видимого лестничного эффекта.

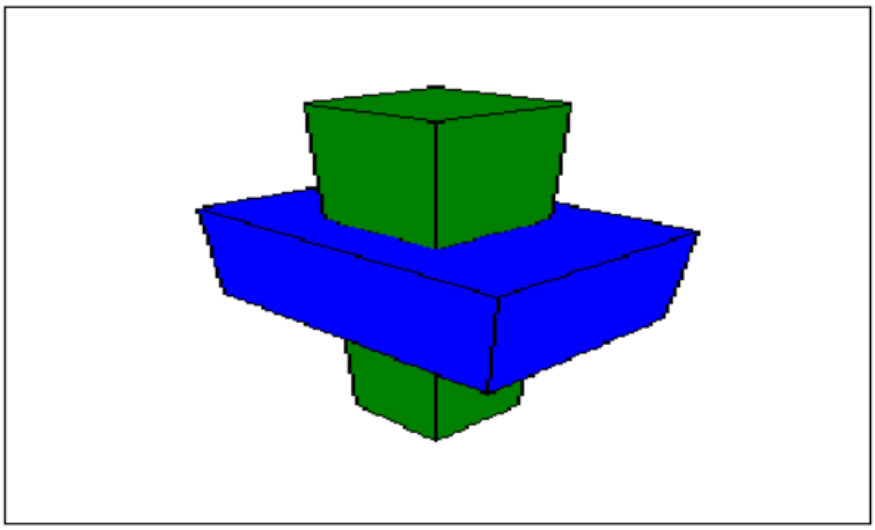


Рисунок 4.32 – Применение алгоритма для тестовой сцены

Алгоритм Z-буфера легко модифицируется для получения сечений поверхности параллельными плоскостями. Для этого на экран выводятся только проекции точек с z-координатой в требуемом интервале.

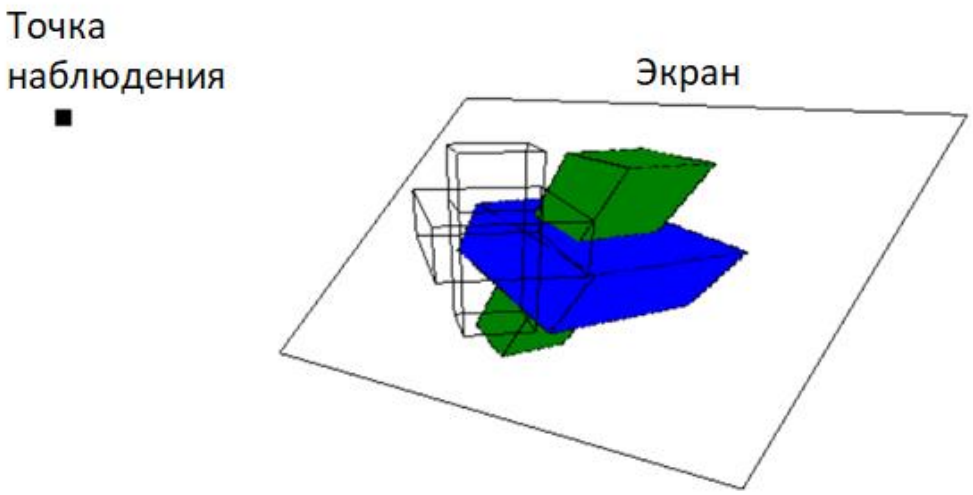


Рисунок 4.33 – Применение алгоритма для тестовой сцены

Сложность алгоритма зависит от числа пикселей и количества объектов: $O(c * n)$, где c – число пикселей, n – число объектов.

Исходный код основных методов класса Z-Buffer представлен в приложении З. Функционал данных методов следующий:

– RemoveHiddenEdges() (листинг З.1) организует проход по всем граням объектов в сцене. Очередная грань проецируется на созданную картинную плоскость, разлагается в растр;
– DrawBresenhamLine() (листинг З.2) реализует логику алгоритма Брезенхема для растеризации отрезков; применяется для каждого ребра рассматриваемой грани;
– Paint() (листинг З.3) с применением сканирующей строки закрашивает определенную область путем сравнения буферов глубины;
– incrementPhase(), decrementPhase(), drawPhase() (листинг З.4) – переопределение методов абстрактного класса, отвечающих за перемещение между этапами алгоритма, их отрисовку.

4.2. РЕАЛИЗАЦИЯ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА

Перед реализацией графического интерфейса зададим несколько сцен и инициализируем их в программе (рисунок 4.34). Основной критерий для новых сцен – большое количество перекрытий и пересечений объектами друг друга. Объектов в каждой сцене не больше трех, т.к. рост их числа увеличивает количество рассматриваемых этапов, что негативно влияет на наглядность демонстрации работы алгоритмов.

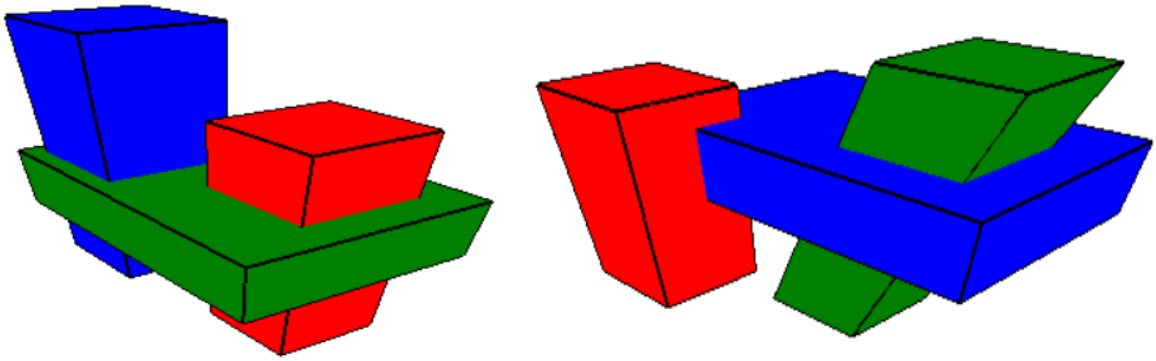
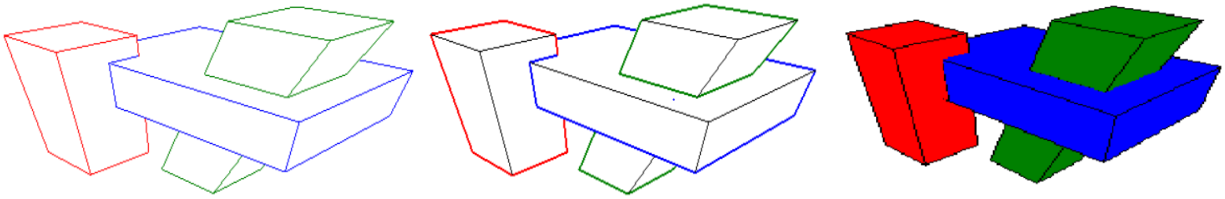


Рисунок 4.34 – Доступные сцены для визуализации алгоритмов

Так, применение алгоритмов для одной из созданных сцен продемонстрировано на рисунке 4.35.



Робертс Аппель Z-буфер

Рисунок 4.35 – Результаты применения алгоритмов для созданной сцены

Реализация графического интерфейса начинается с задания стартовой формы MenuForm (рисунок 4.36), которая предоставляет пользователю основные действия доступные из приложения: изучение теоретического материала и запуск визуализатора. Данные действия доступны по нажатии специальных ссылок, которые реализованы с помощью инструмента LinkLabel. Для удобства компоненты верхнего меню настроек повторяют функционал основных действий. Компонент «Справка», при необходимости, может использоваться как первичное ознакомление с программным комплексом. Для создания более отзывчивого интерфейса на форме содержится краткое описание и иллюстрация необходимости алгоритмов загораживания.

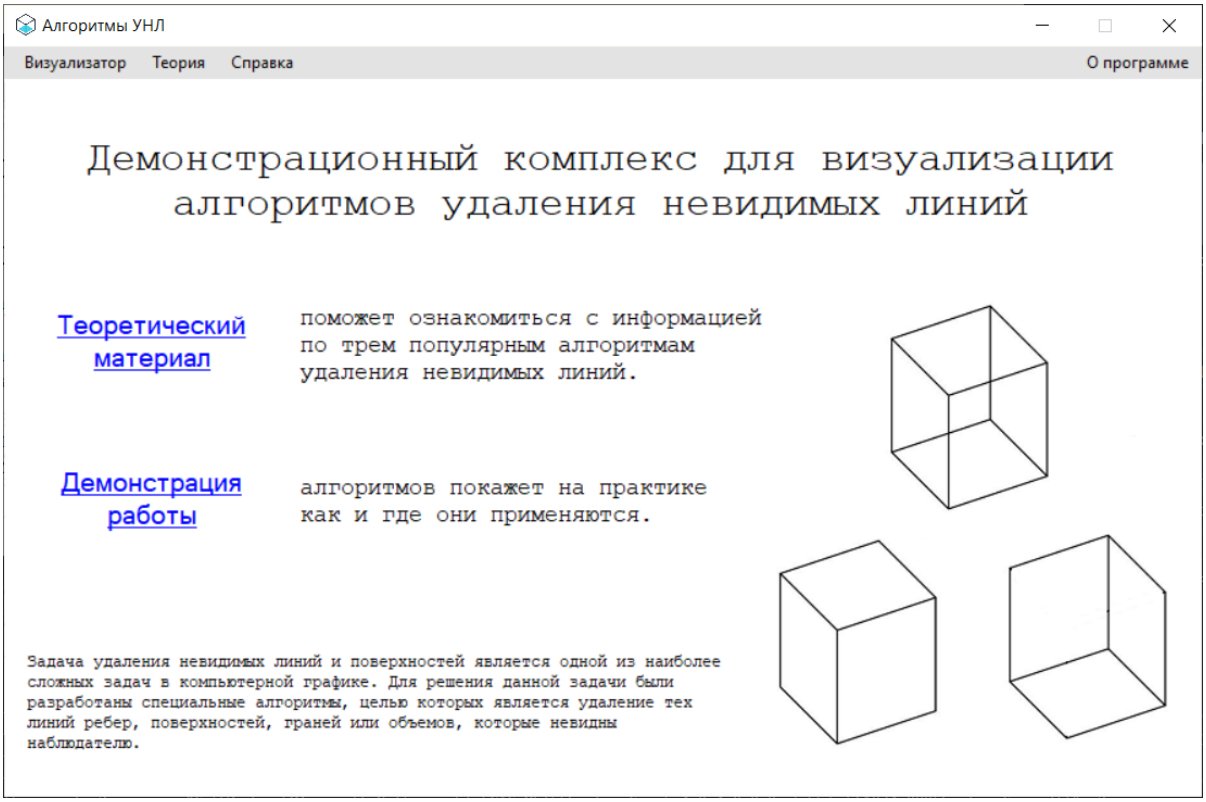


Рисунок 4.36 – Стартовое меню приложения

Перед демонстрацией работы алгоритмов пользователю необходимо ознакомиться с описанием их работы. Материал для этого был взят из учебно-

методического пособия «Методы и средства представления графической информации» автора к.т.н. Ярош Е.С. Информация была структурирована и разделена на отдельные файлы формата *rtf*. Важным преимуществом *rtf* файла является совместимость с многочисленными операционными системами и приложениями для работы с текстом, что может быть актуально, если пользователь запустит ДПК на другой версии Windows.

После запуска окна «Теоретический материал» в специальный компонент для работы с текстом RichTextBox методом LoadFile() загружается файл, содержащий общую информацию по алгоритмам удаления невидимых линий (рисунок 4.37). После выбора необходимого алгоритма в верхнем меню будет загружен определенный файл с описанием работы алгоритма.

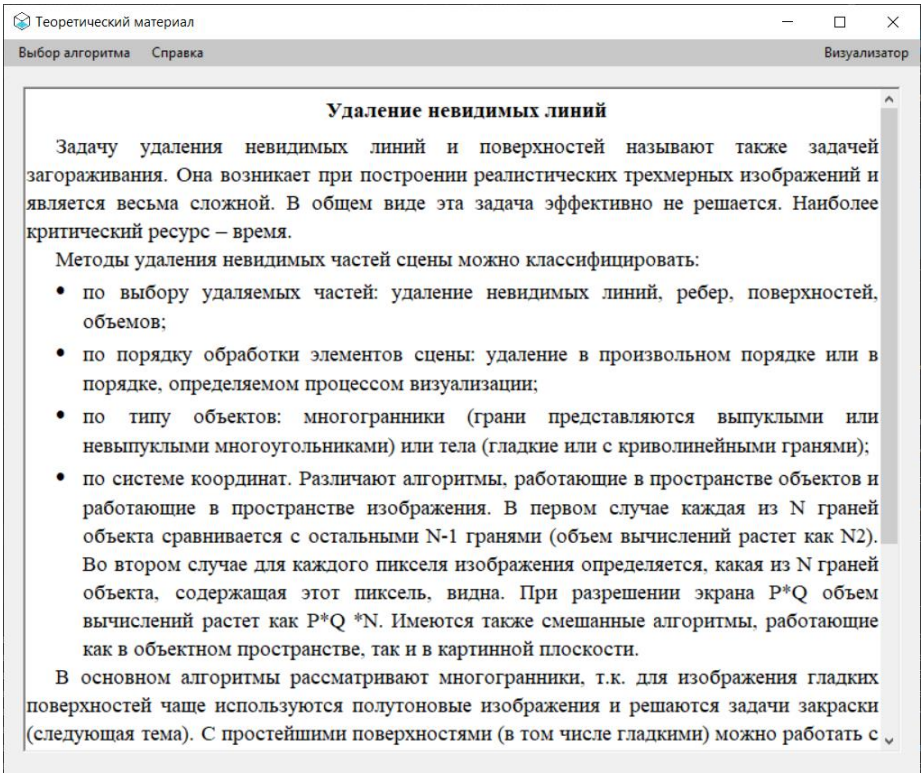


Рисунок 4.37 – Форма для изучения теоретического материала

Так как реализация доступа к теоретическому материалу перед началом работы была не основным требованием, а скорее дополнительным, данное окно

было разработано в простейшем виде с минимальным количеством функциональных возможностей.

Если пользователь уверен в своих знаниях, то он сразу из стартового меню может перейти к окну с демонстрацией работы алгоритмов (рисунок 4.38). Из окна с теорией это действие можно выполнить, выбрав компонент Визуализатор из верхнего меню настроек.

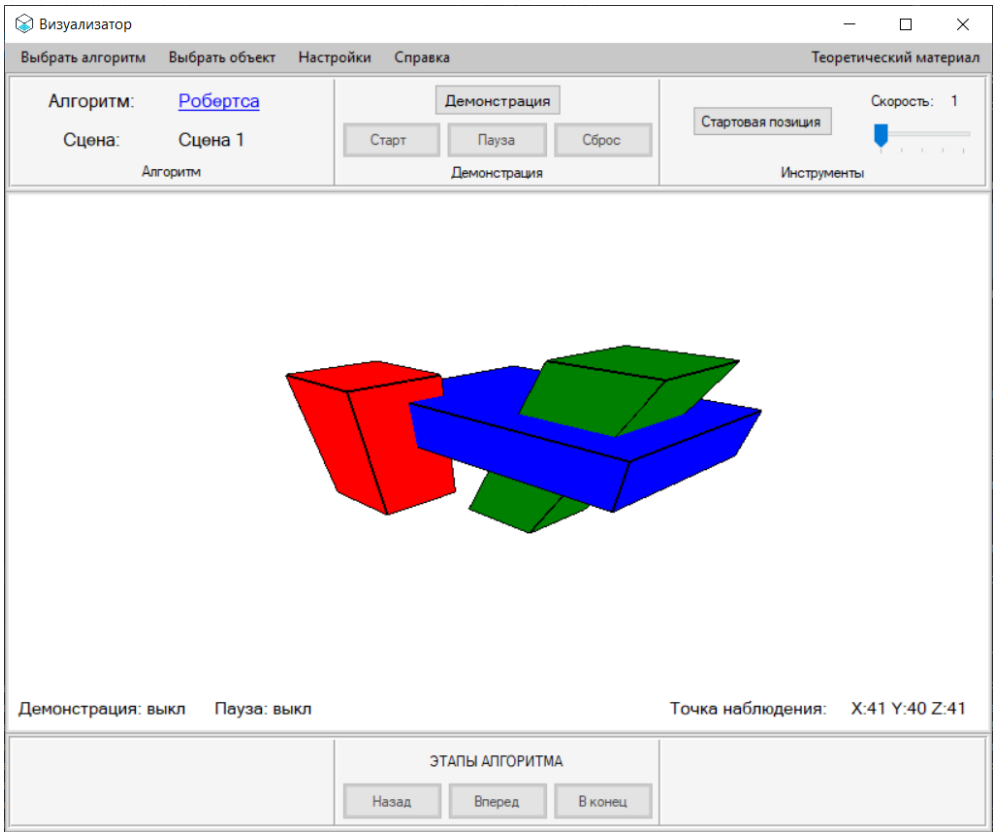


Рисунок 4.38 – Окно демонстрации работы алгоритмов

После запуска окна происходит отрисовка всех компонентов формы, в том числе компонента glControl, в котором сразу отображается одна из сцен. В целом пользователю доступны следующие действия:

1. По нажатии кнопки «Демонстрация работы» становится доступна группа кнопок, отвечающих за манипулирование процессом визуализации.

a. «Старт» отвечает за запуск процесса визуализации. После ее нажатия рисуются все ребра каждого из объектов в сцене. От этапа к этапу выбирается тестовое ребро или грань, выделяются необходимые элементы в зависимости от выбранного алгоритма;
b. «Пауза» отвечает за остановку процесса визуализации. По нажатии кнопки становится доступная группа кнопок, отвечающая за перемещение между этапами хода работы алгоритмов.

2. Группа кнопок «Этапы алгоритма» позволяет перемещаться между этапами вперед, назад, а также в конец, т.е. перейти к последнему этапу, после которого применение алгоритма заканчивается.
3. Во время визуализации алгоритма пользователь может изменять положение точки наблюдения для лучшего рассмотрения всех деталей. Кнопка «Стартовая позиция» отвечает за возвращение точки наблюдения в стартовое положение.

Пользователю также доступен некоторый набор дополнительных инструментов, которые могут облегчить взаимодействие с процессом визуализации алгоритмов (рисунок 4.39).

1. Позиционирование объема видимости в пространстве, т.е. установка камеры в каких-либо координатах.
2. Изменение скорости визуализации. Данное действие происходит с помощью специального компонента в верхнем правом углу окна. Доступно 5 скоростей, изначально установлена первая – самая медленная. Заданная скорость влияет на визуализацию только до нажатия кнопки «Пауза», т.е. до перехода в ручное управление с помощью специальных кнопок.

3. Перемещение между этапами путем задания номера этапа. Специальный компонент в нижнем правом углу окна позволяет указать числовое значение, соответствующее нужному этапу.
4. Информация о текущей алгоритме, а также текущей выбранной сцене отображается с помощью компонентов Lable в верхнем левом углу окна визуализатора. По нажатии на название алгоритма происходит переход к окну с теоретическим материалом по данному алгоритму.
5. Дополнительная информация, в том числе о положении точки наблюдения, а также о текущем состоянии кнопок «Демонстрация» и «Пауза» отображается ниже панели glControl.

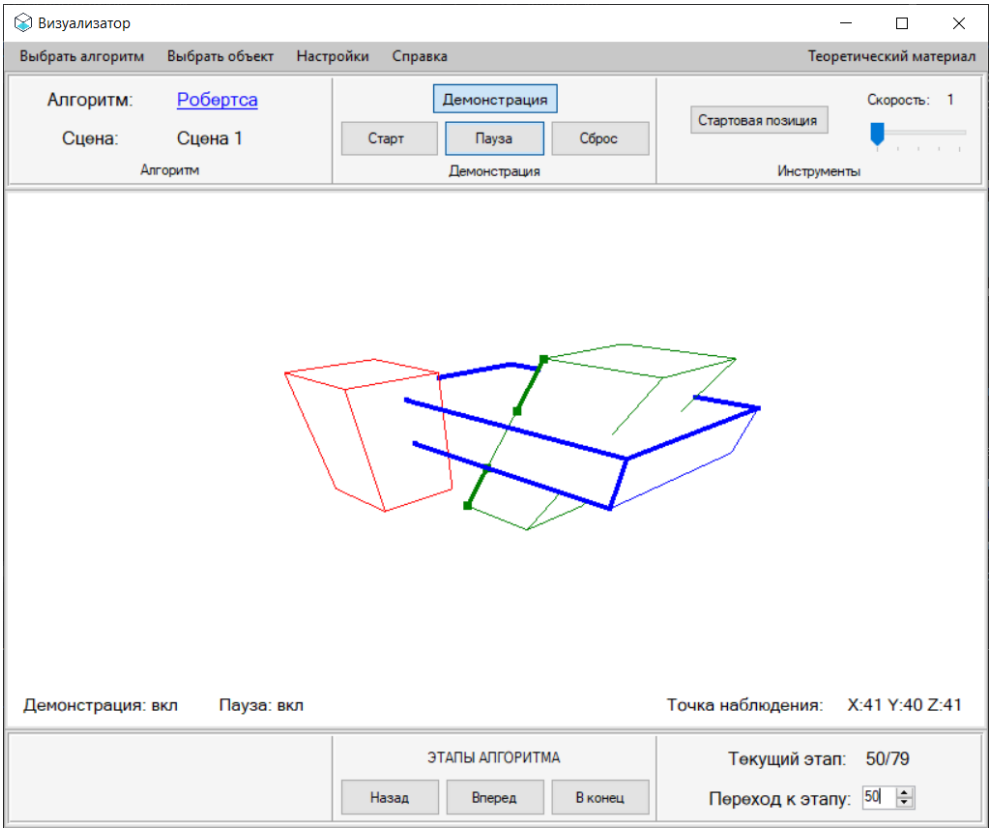


Рисунок 4.39 – Доступные инструменты управления

С помощью верхнего меню пользователь может выбрать нужный ему алгоритм, а также сцену для визуализации. Выбор какого-либо варианта

очищает информацию уже присутствующую на экране и возвращает сцену в ее первоначальное состояние.

Помимо выбора алгоритма и сцены в верхнем меню находится доступ к окну с настройками (рисунок 4.40). После его открытия на каждой из панелей в ряд располагаются элементы, которые представляют из себя параметры визуализации. Задавая их, пользователь настраивает процесс визуализации. Например, параметры для Z-буфера позволяют настроить отображение пространственного объекта, а также разрешение экрана.

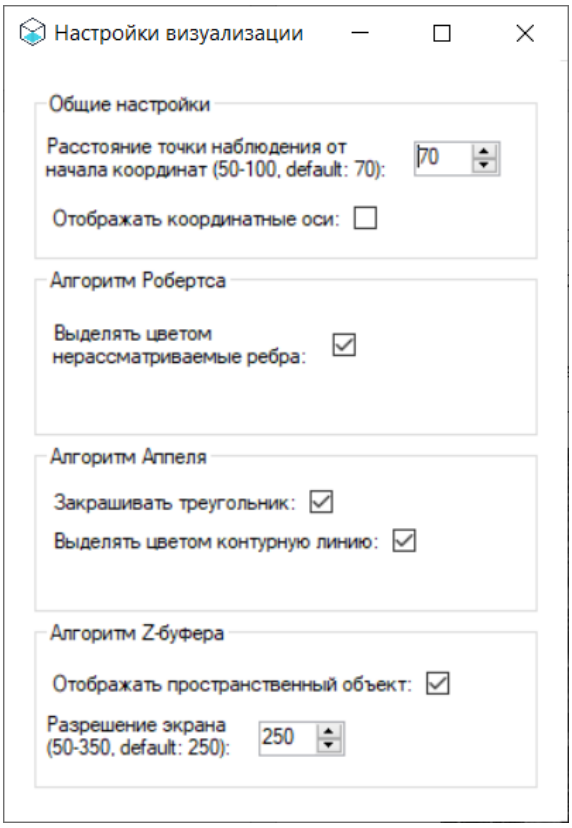


Рисунок 4.40 – Окно с настройками визуализации

Все настройки изменяются в реальном времени и носят визуальный эффект. Одной из опций является отрисовка координатных осей в сцене (рисунок 4.41). Знание пользователем положение объекта относительно начала координат не является обязательным при демонстрации алгоритмов, т.к.

основная информация заключается в положении объекта относительно точки наблюдения. Таким образом, отрисовка координатных осей является опциональной функцией.

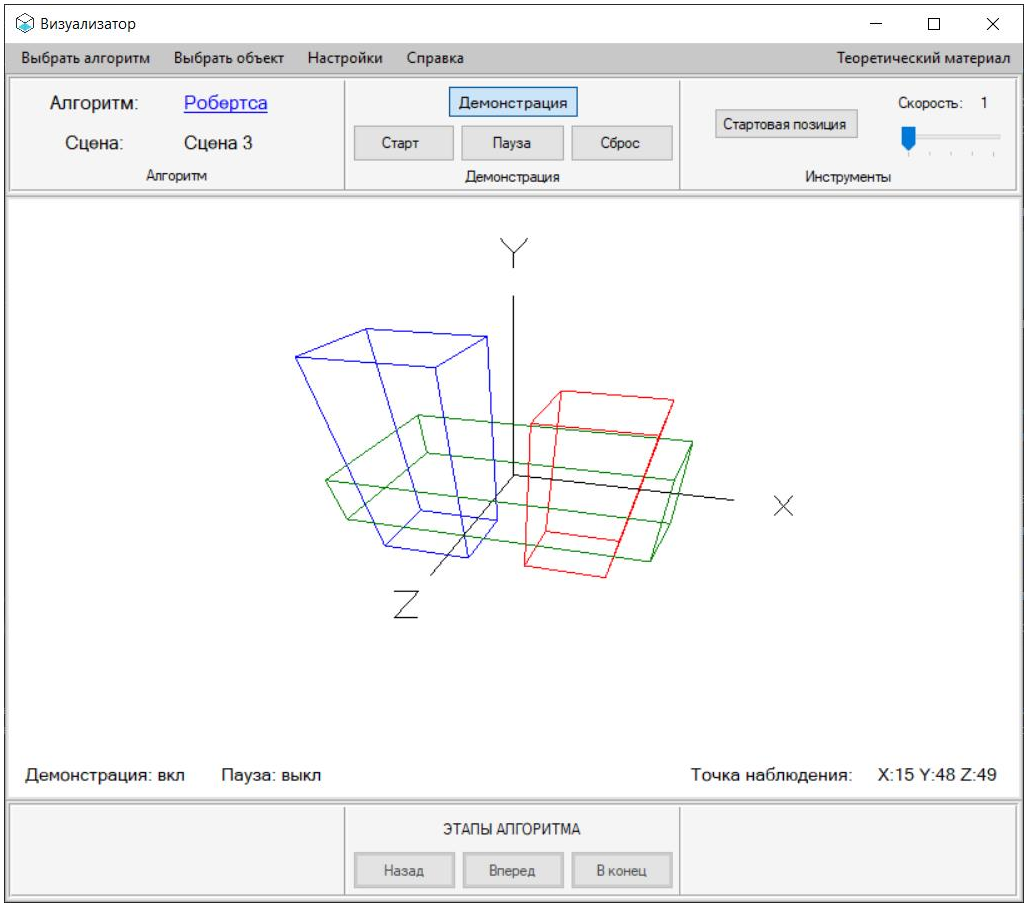


Рисунок 4.41 – Использование координатных осей

Для алгоритма Z-буфера (рисунок 4.42) важной частью является расчет буферов глубины и кадра. Кнопки доступа к буферам становятся доступны после старта визуализации алгоритма Z-буфера. После нажатия одной из них открывается специальная форма, в которой отображается содержимое выбранного буфера, изменяемое от этапа к этапу.

На рисунках 4.43, 4.44 представлены буферы кадра для конечных этапов применения алгоритма с разрешением экрана 50×50 и 250×250. Значения, отличающиеся от значений фона, выделены цветом соответствующего объекта.

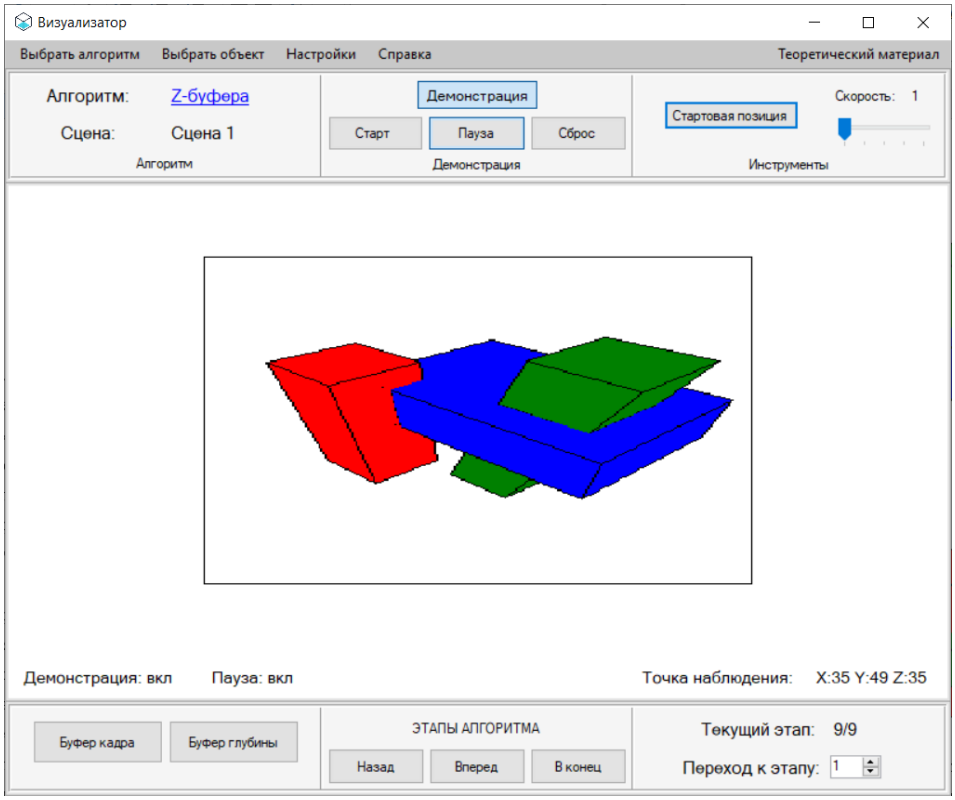


Рисунок 4.42 – Демонстрация алгоритма Z-буфера

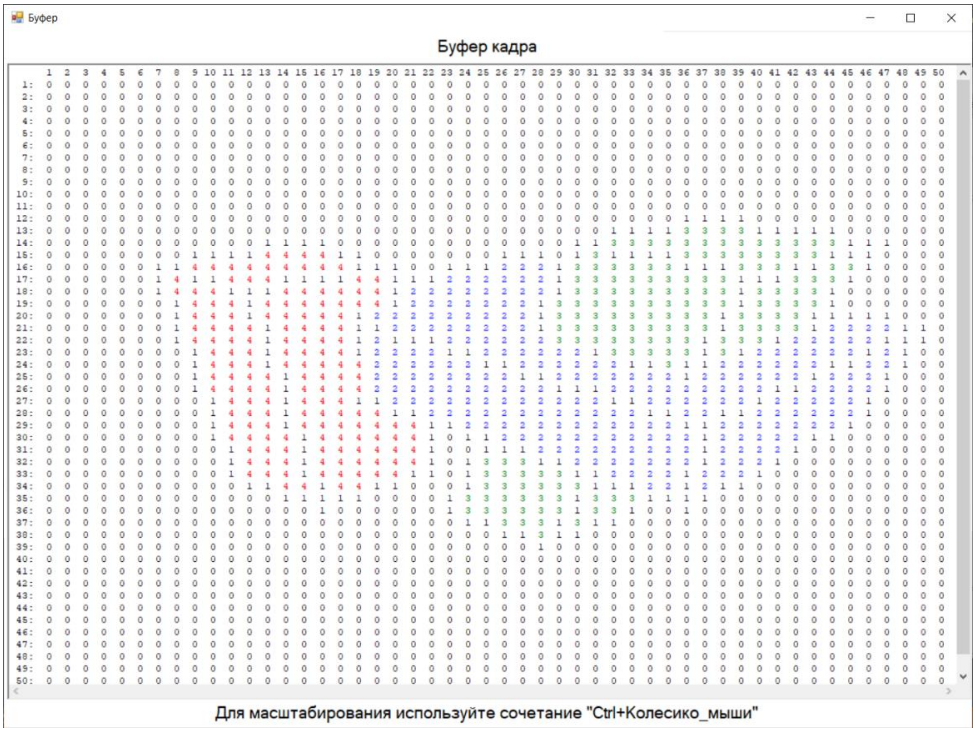


Рисунок 4.43 – Демонстрация алгоритма Z-буфера. Экран 50×50

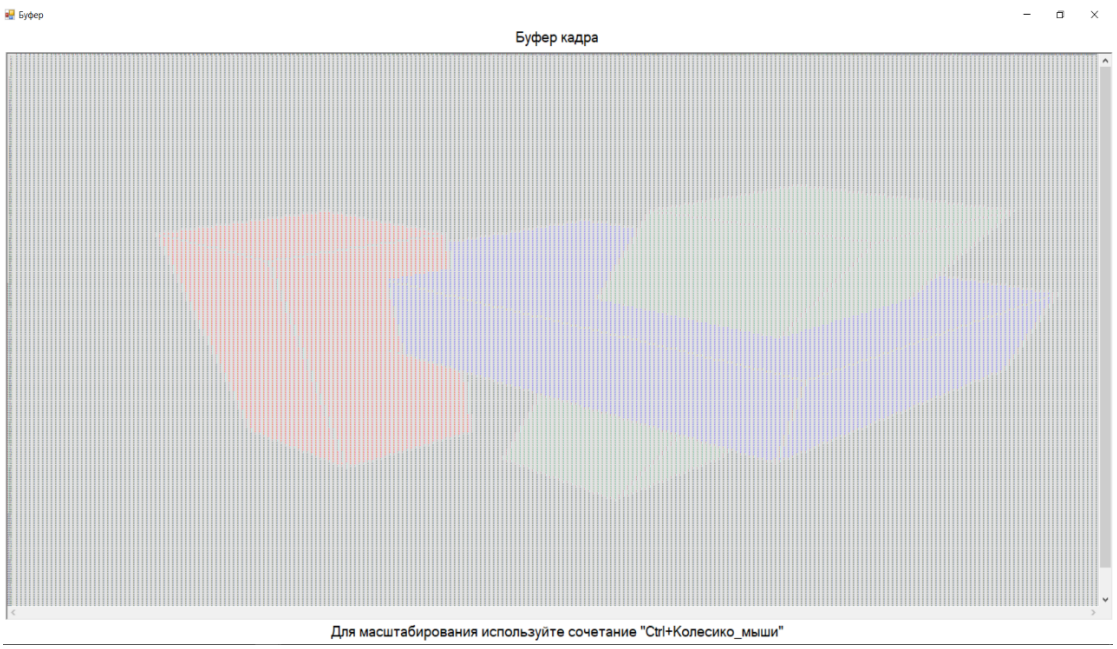


Рисунок 4.44 – Демонстрация алгоритма Z-буфера. Экран 250×250

Важной частью демонстрации применения алгоритмов является возможность вращения всей сцены, что в свою очередь происходит только с изменением точки наблюдения. Выполнять это действие удобнее всего с помощью мыши. Так, в процессе работы программы высчитывается очередное перемещение мыши по окну, преобразуется в соответствующие углы поворота по вертикали и горизонтали. Следующим шагом изменяются соответствующие свойства объекта viewPoint класса ViewPoint, отвечающие за углы от оси X, и Y. Внутри методов класса ViewPoint определяются новые координаты камеры, создаётся новая матрица ModelView:

G_modelview = Matrix4.LookAt(vp.viewX, vp.viewY, vp.viewZ, 0, 0, 0, 0, vp.orintation_y, 0), где:

– первая группа из 3 элементов задает координаты камеры;
– вторая группа из 3 элементов задает направление камеры;
– последние 3 элемента задают поворот камеры, т.е. направление оси *Y* вверх, либо вниз.

Все изменения будут обрабатываться в методах MouseDown(), MouseMove(), MouseUp().

Особенности разработанного графического интерфейса:

– интерфейс ДПК выполнен в классическом стиле приложений Windows Forms в нейтральном сером цвете и его оттенках, для фона и текста выбраны контрастные цвета;
– при изменении размеров окна данные на визуализаторе сохраняются, что в свою очередь реализовано обработкой события Resize;
– при возникновении каких-либо ошибок в вычислениях обрабатываются исключения, пользователь получает информационное окно с указанием ошибки.

Исходный код основных методов класса формы VisualisatorForm представлен в приложении И. Функционал этих методов следующий:

– glControlMain_Load обрабатывает событие Load при загрузке компонента glControl;
– glControlMain_Paint обрабатывает событие Paint, вызываемое при изменениях, влияющих на компонент glControl;
– Display() обрабатывает текущую сцену, последовательно отрисовывает все ребра фигур после применения какого-либо алгоритма, учитывая его особенности;
– обработчики нажатий кнопок реализуют основную логику взаимодействия пользователя с графическим интерфейсом.

5. ТЕСТИРОВАНИЕ

Разработка проекта велась исключительно в ходе дипломного проектирования, поэтому на данный момент было проведено только альфа-тестирование. Альфа-тестирование – имитация реальной работы с системой разработчиком, либо реальная работа с системой потенциальными пользователями. Тестирования проводились в процессе разработки, а также для законченного продукта в качестве внутреннего приёмочного тестирования. В ходе тестирования принималась отладочная информация, выводимая в консоль, делались необходимые поправки.

Разработанное десктопное приложение для системы Windows размещается на компьютере пользователя. Для работы оно не требуют подключение к интернету, взаимодействует с пользователем посредством стандартного интерфейса, имеет высокое быстродействие. Приложение зависит от используемой операционной системы, т.к. не является кроссплатформенным, требует установку на каждый компьютер пользователя, желающего работать с данным приложением.

Заказчиком было проведено тестирование с целью определения удобен ли пользовательский интерфейс для его предполагаемого применения, а также весь ли требуемый функционал реализован. Тестирование прошло успешно, визуализация происходила без ошибок, отклик от всех компонентов был быстрый. Все пожелания были своевременно добавлены в ДПК, произведена программная оптимизация.

Тестирование для десктопного приложения на предмет ошибок инсталляции, обновления, деинсталляции не проводился, т.к. приложение предоставляется не в сжатом формате.

6. ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было спроектировано и разработано десктоп-приложение «Демонстрационный программный комплекс для изучения алгоритмов загораживания Робертса, Аппеля, Z-буфера».

В ходе выполнения работы решены следующие задачи:

1. Обоснована актуальность разработки.
2. Проведен обзор аналогов.
3. Выбраны инструменты разработки.
4. Спроектирована программная система.
5. Выполнена реализация проекта.

В приложение включены наиболее распространенные алгоритмы загораживания:

– алгоритм Робертса;
– алгоритм Аппеля;
– алгоритм Z-буфера.

Для того, чтобы определить требования к программной системе, была построена модель прецедентов с описанием функционала. По выделенным подсистемам демонстрационного комплекса было проведено описание основных модулей системы, построены диаграммы ее компонентов.

На языке программирования C# было разработано приложение, демонстрирующее применение алгоритмов загораживания, а также произведена интеграция теоретического материала в приложение для более подробного изучения пользователем предметной области. Для работы с графикой использовалась внешняя библиотека OpenTK (OpenGL 4.0). Готовый

демонстрационный комплекс позволяет проследить все стадии изменения объекта при применении к нему каждого из алгоритмов.

Предусмотрено задание и сохранение настроек визуализации (положение точки наблюдения, выделение элементов цветом), которые в последующем запуске программы восстанавливаются. При ошибках вычислений обрабатываются исключения, пользователь получает информационное окно с указанием ошибки.

Преимущества разработанного ДПК перед найденными решениями:

– материал структурирован, сопровождается подробным объяснением;
– алгоритмы располагаются в одном месте, что обеспечивает удобство и исключает затраты на поиск;
– предоставление настроек визуализации, а именно ввод исходных данных, изменение скорости визуализации, изменение положения точки наблюдения;
– гарантия работоспособности и безопасности.

Разработка приложения закончена и передана заказчику.

Полный исходный код ДПК представлен в репозитории GitHub [21].

Некоторые перспективы развития: данный продукт можно доработать до универсального учебного пособия по машинной графике, добавив другие темы для изучения (например, закрашивание, отсечение отрезков).

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ярош, Е.С. Методы и средства представления графической информации: учебное пособие / Е.С. Ярош. – Челябинск: Издательский центр ЮУрГУ, 2017. – 230 с.
2. Митин, А.И. Компьютерная графика: справочно-методич. пособие. / А.И. Митин, Н.В. Свертилова – 2-е изд., стереотип. – М.-Берлин: Директ-Медиа, 2016. – 251 с.
3. Удаление невидимых линий и поверхностей. – http://compgraph.tpu.ru/Del_hide_line.htm. Дата обращения: 15.02.2020.
4. Алгоритм Робертса. – http://compgraph.tpu.ru/roberts.htm. Дата обращения: 01.03.2020.
5. Алгоритм, использующий Z-буфер. – http://compgraph.tpu.ru/zbuffer.htm. Дата обращения: 01.03.2020.
6. Реализация алгоритма Робертса. – http://algolist.ru/graphics/roberts.php. Дата обращения: 01.03.2020.
7. Set of small graphics programs fully written on C++. – https://github.com/riz-f/graphics-programs. Дата обращения: 04.03.2020.
8. The problem of hidden surface removal. – http://learnwebgl.brown37.net/11_advanced_rendering/hidden_surface_removal.html. Дата обращения: 04.03.2020.
9. Hidden Surface Removal with Java. – https://github.com/gioflores24/hidden-surface-removal . Дата обращения: 04.03.2020.
10. Hidden surface removal using the painter’s algorithm. – https://github.com/Skar0/sddII. Дата обращения: 04.03.2020.

11. Operating System Market Share Worldwide. – https://gs.statcounter.com/os-market-share. Дата обращения: 15.03.2020.
12. Сравнительный тест DirectX 11, OpenGL и Vulkan. – https://www.hardwareluxx.ru/index.php/artikel/hardware/grafikkarten/37520-directx-11-vulkan-opengl.html. Дата обращения: 16.03.2020.
13. OpenTK. – https://opentk.net/faq.html. Дата обращения: 25.03.2020.
14. Learn OpenGL. Hello Window. – https://learnopengl.com/Getting-started/Hello-Window. Дата обращения: 25.03.2020.
15. Learn OpenGL. Coordinate System. – https://learnopengl.com/Getting-started/Coordinate-Systems. Дата обращения: 25.03.2020.
16. OpenGL. – http://opengl-tutorial.blogspot.com/p/3.html. Дата обращения: 26.03.2020.
17. Roberts, L.G. Machine perception of three-dimensional solids. / L.G. Roberts. – Department of Electrical Engineering. Massachusetts Institute of Technology, 1963. – 82 с.
18. Appel, A. The Notion of Quantitive Invisibility and the Machine rendering of Solids. / A. Appel. – Proceedings of ACM National Meeting, 1967. – 387 с.
19. Боресков, А.В. Программирование компьютерной графики. Современный OpenGL. / А.В. Боресков. – М.: ДМК Пресс, 2019. – 372 с.
20. Козлова, А.В. Разработка демонстрационного комплекса программ для изучения алгоритмов растеризации: выпускная квалификационная работа / А.В. Козлова. – ЮУРГУ, ВШЭКН, Каф. ЭВМ. – Челябинск, 2018. – 76 с.
21. Росс, К.А. Исходный код демонстрационного комплекса. – https://github.com/gospella/hsr-algorithms. Дата обращения: 31.05.2020.