Министерство образования и науки 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

Зав. кафедрой

_____ Г.И. Радченко

«___»_____2018г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к выпускной квалификационной работе

Тема: «Разработка демонстрационного комплекса для изучения алгоритмов загораживания»

ЮУрГУ 090301.2018.153 ПЗ ВКР

Руководитель

_____/доц. Е.С. Ярош

«___»_____2018г.

Автор работы

студент группы КЭ-484

_____/А.Р. Валиахмедов

«___»_____2018 г.

Нормоконтроллер,

ст. преп. ЭВМ ВШЭКМ

_____/ В.В. Лурье

«___» _____2018 г.

Челябинск 2018

АННОТАЦИЯ

Валиахмедов А.Р., Разработка демонстрационного комплекса для изучения алгоритмов загораживания: Пояснительная записка к выпускной квалификационной работе. – Челябинск: ЮУрГУ, КЭ - 484, 59с., 28 илл., библиогр. список – 21 наим.

Выпускная квалификационная работа посвящена разработке программного обеспечения, которое наглядным образом демонстрирует работу методов и алгоритмов удаления невидимых линий и поверхностей.

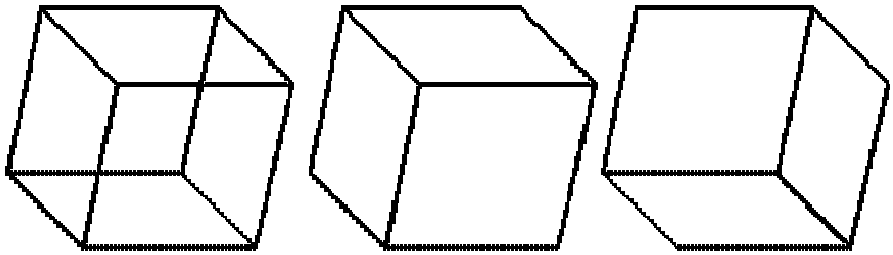
После анализа существующей информации по курсу компьютерной графики в разделе удаления невидимых линий и поверхностей выявлено, что присутствует теоретическая основа всех методов и алгоритмов. Она объясняет суть работы методов и алгоритмов, но не демонстрирует наглядного представления их работы. Существуют аналоги: алгоритм плавающего горизонта и метод Z буфера. Они демонстрируют ход работы методов и алгоритмов, но не дают теоретическую основу.

Вследствие этого был разработан демонстрационный комплекс программ, который собирает в себе теоретическую основу, схемы, описывающие алгоритмы и методы, и наглядное представление их хода работы. Комплекс программ направлен на понимание студентами методов и алгоритмов удаления невидимых линий и поверхностей.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в машинной графике. 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. Необходимость удаления невидимых линий, ребер, поверхностей или объемов проиллюстрирована рисунке 1.



а б в

Рисунок 1 - Необходимость удаления невидимых линий

На рисунке 1, а приведен типичный каркасный чертеж куба. Каркасный чертеж представляет трехмерный объект в виде штрихового изображения его ребер. Рисунке 1, а можно интерпретировать двояко: как вид куба сверху слева или снизу справа. Удаление тех линий или поверхностей, которые невидимы с соответствующей точки зрения, позволяют избавиться от неоднозначности. Результаты показаны на рисунке 1, б и 1, в.

Сложность задачи удаления невидимых линий и поверхностей привела к появлению большого числа различных способов ее решения. Многие из них ориентированы на специализированные приложения. Наилучшего решения обшей задачи удаления невидимых линий и поверхностей не существует. Для моделирования процессов в реальном времени, например для авиатренажеров, требуются быстрые алгоритмы, которые могут порождать результаты с частотой видеогенерации 30 кадр/с. Для машинной мультипликации, например, требуются алгоритмы, которые могут генерировать сложные реалистические изображения, в которых представлены тени, прозрачность и фактура, учитывающие эффекты отражения и преломления цвета в мельчайших оттенках. Подобные алгоритмы работают медленно и зачастую на вычисления требуется несколько минут или даже часов. Строго говоря, учет эффектов прозрачности, фактуры, отражения и т. п. не входит в задачу удаления невидимых линий или поверхностей. Естественнее считать их частью процесса визуализации изображения. Однако многие из этих эффектов встроены в алгоритмы удаления невидимых

поверхностей. Существует тесная взаимосвязь между скоростью работы алгоритма и детальностью его результата. Ни один из алгоритмов не может достигнуть хороших оценок для этих двух показателей одновременно. По мере создания все более быстрых алгоритмов можно строить все более детальные изображения. Реальные задачи, однако, всегда будут требовать учета еще большего количества деталей.

Все алгоритмы удаления невидимых линий (поверхностей) включают в себя сортировку. Порядок, в котором производится сортировка координат объектов, вообще говоря, не влияет на эффективность этих алгоритмов. Главная сортировка ведется по геометрическому расстоянию от тела, поверхности, ребра или точки до точки наблюдения. Основная идея, положенная в основу сортировки по расстоянию, заключается в том, что чем дальше расположен объект от точки наблюдения, тем больше вероятность, что он будет полностью или частично заслонен одним из объектов, более близких к точке наблюдения. После определения расстояний или приоритетов по глубине остается провести сортировку по горизонтали и по вертикали, чтобы выяснить, будет ли рассматриваемый объект действительно заслонен объектом, расположенным ближе к точке наблюдения. Эффективность любого алгоритма удаления невидимых линий или поверхностей в большой мере зависит от эффективности процесса сортировки. Для повышения эффективности сортировки используется также когерентность сцены, т. е. тенденция неизменяемости характеристик сцены в малом.

Алгоритмы удаления невидимых линий или поверхностей можно классифицировать по способу выбора системы координат или пространства, в котором они работают. Выделяют три класса алгоритмов удаления невидимых линий или поверхностей:

- алгоритмы, работающие в объектном пространстве.
- алгоритмы, работающие в пространстве изображения (экрана).
- алгоритмы, формирующие список приоритетов.

Алгоритмы, работающие в объектном пространстве, имеют дело с физической системой координат, в которой описаны эти объекты. При этом получаются весьма точные результаты, ограниченные лишь точностью вычислений. Полученные изображения можно свободно увеличивать во много раз. Алгоритмы, работающие в объектном пространстве, особенно полезны в тех приложениях, где необходима высокая точность.

Алгоритмы, работающие в пространстве изображения, имеют дело с системой координат того экрана, на котором объекты визуализируются. При этом точность вычислений ограничена разрешающей способностью экрана.

Результаты, полученные в пространстве изображения, а затем увеличенные во много раз, не будут соответствовать исходной сцене. Например, могут не совпасть концы отрезков. Алгоритмы, формирующие список приоритетов, работают попеременно в обеих упомянутых системах координат.

Объем вычислений для любого алгоритма, работающего в объектном пространстве и сравнивающего каждый объект сцены со всеми остальными объектами этой сцены, растет теоретически как квадрат числа объектов (n^2). Аналогично, объем вычислений любого алгоритма, работающего в пространстве изображения и сравнивающего каждый объект сцены с позициями всех пикселей в системе координат экрана, растет теоретически, как n в степени N. Здесь n обозначает количество объектов (тел, плоскостей или ребер) в сцене, а N — число пикселей. Теоретически трудоемкость алгоритмов, работающих в объектном пространстве, меньше трудоемкости алгоритмов, работающих в пространстве изображения, при $n < N$. Однако на практике это не так. Дело в том, что алгоритмы, работающие в пространстве изображения, более эффективны потому, что для них легче воспользоваться преимуществом когерентности при растровой реализации.

1 ОБЗОР И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Описание проблемы

Компьютерная графика является сложной областью деятельности, при изучении которой возникает множество трудностей в понимании методов и алгоритмов. Студенту нередко приходится самостоятельно продумывать, как происходит выполнение того или иного метода или искать наглядное пособие. Зачастую таких пособий нет.

Поэтому если собрать в одном комплексе программ и теоретическую часть, и практическую, а также добавить блок-схемы алгоритмов, то понимание студентами процесса работы графических алгоритмов будет более полным.

Наглядное представление работы алгоритмов и методов может помочь не только студентам, но и преподавателям. На визуальных примерах учащиеся могут лучше разобрать, как устроен метод или алгоритм, а также, как он выполняется и какие вычисления проводятся по мере работы алгоритма. Преподаватели могут представить студентам блок-схему алгоритма и лучше донести до них читаемый материал.

Главным достоинством наглядного метода представления данных является его поэтапность, то есть выполнение метода или алгоритма происходит постепенно, и студент успевает понять и усвоить суть его работы.

1.2 Существующие решения

Поиск программных комплексов для демонстрации всех требуемых алгоритмов оказался безрезультатным. Существуют лишь примеры отдельных алгоритмов (алгоритм плавающего горизонта и метод Z буфера). Данные алгоритмы можно использовать для обучения. Но они не дают комплексного понимания всего раздела удаления невидимых линий и поверхностей.

Также существует теоретическое описание методов и алгоритмов. Алгоритм Робертса и метод Z буфера[5], метод сортировки по глубине, метод Z буфера, метод построчного сканирования[16], алгоритм Аппеля [17].

1.3 Достоинства и недостатки существующих решений

В таблице 1. приведены некоторые найденные универсальные движки.

Таблица 1 - Преимущества и недостатки имеющихся решений

Наименование	Преимущества	Недостатки
1. Теоретические материалы	• Подробное описание методов и алгоритмов; • Описывает все методы и алгоритмы удаления невидимых линий.	• Отсутствие визуального представления.
2. Программа для описания метода Z буфера[20]	• Визуальное представление; • Совместимость с различными платформами.	• Отсутствие теоретической базы.
3. Программа для описания алгоритма плавающего горизонта [20]	• Визуальное представление; • Совместимость с различными платформами.	• Отсутствие теоретической базы.

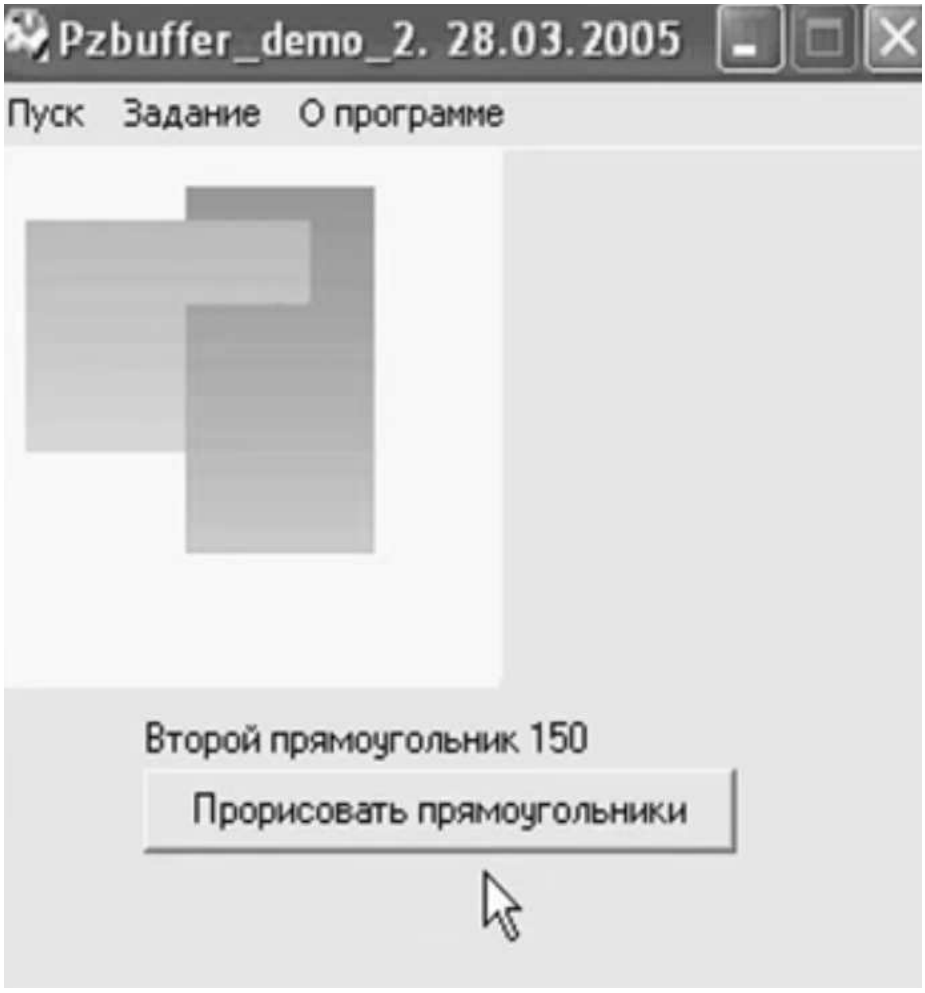


Рисунок 1.1 - Пример работы программы метода Z буфера

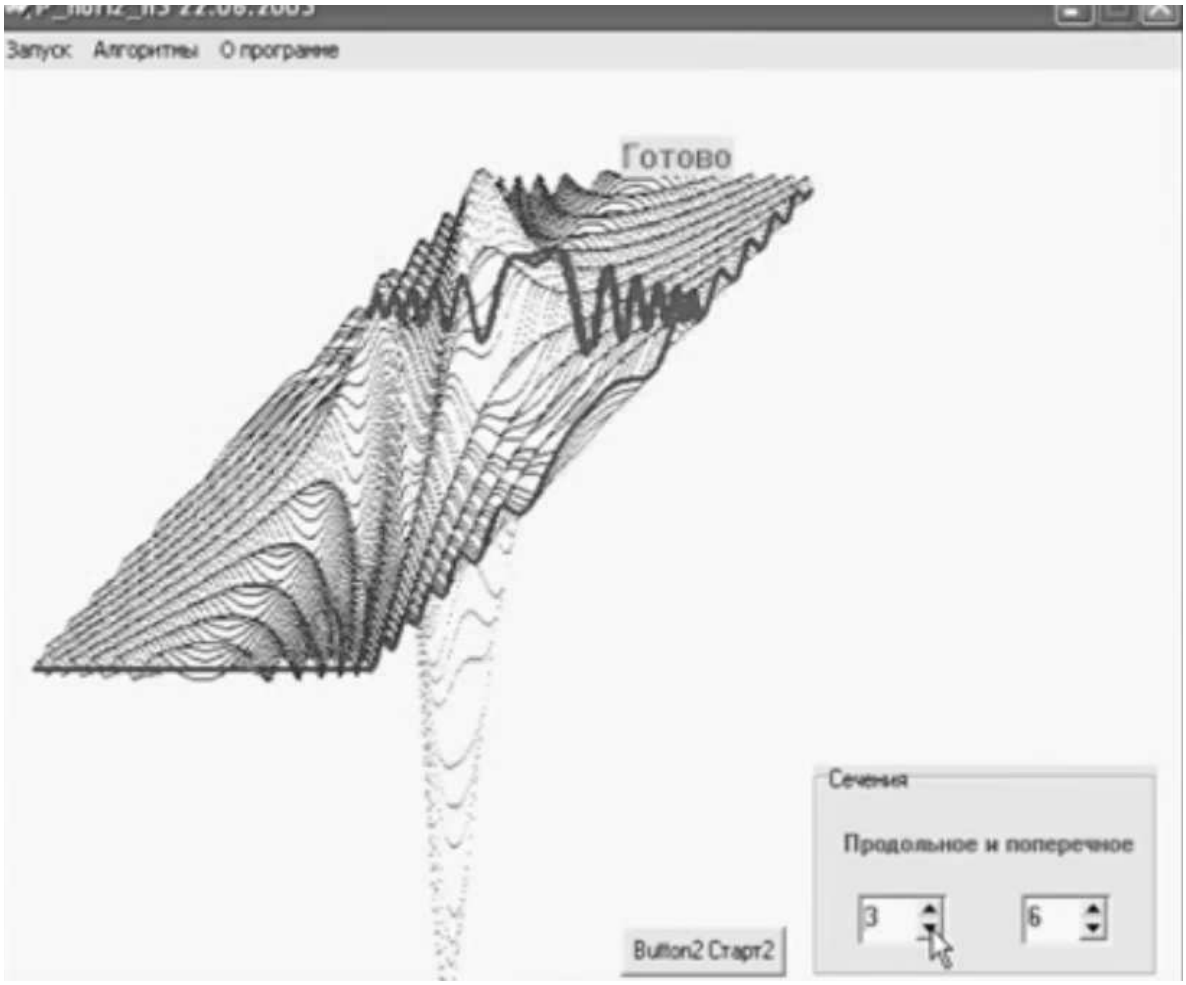


Рисунок 1.2 - Пример работы программы алгоритм плавающего горизонта

Описание представленных программ расположено сайте на http://haidarovg.narod.ru/, однако данный источник не является надежным. Программы нуждаются в проверке на безопасность и корректность выполнения.

Исходя из вышеописанного, необходимо разработать программный комплекс, с помощью которого можно было бы упростить обучение студентов предмету компьютерной графики.

1.4 Достоинство и недостатки нового решения

Преимущества использования нового решения перед найденными решениями из пункта 1.3:

- охват всех изучаемых методов удаления невидимых линий и поверхностей;
- наличие блок-схем для каждого алгоритма;
- наличие теоретической информации по каждому методу;
- визуальное представление всех рассмотренных методов.

Недостатками являются:

- «заточенность» комплекса программ под платформу Windows (в соответствии с заданием).

2 Теоретическое описание программы

Полный список того, что было разработано:

- доступные алгоритмы и методы по удалению невидимых линий и поверхностей для многогранника:

  - алгоритм Робертса;

  - алгоритм Аппеля;

  - метод сортировки по глубине;

  - метод Z-буфера;

  - метод построчного сканирования;

- главное меню с выбором алгоритма для изучения – после запуска демонстрационного комплекса программ для изучения алгоритмов загораживания открывается главное меню, где можно сделать выбор нужного алгоритма;

- теоретический материал по выбранному алгоритму – в главном меню демонстрационного комплекса программ для изучения алгоритмов загораживания можно выбрать изучаемый алгоритм, по нажатию кнопки откроется окно с теоретическим материалом по выбранному алгоритму;

- просмотр блок-схемы по выбранному алгоритму – окно с блок-схемами открывается после нажатия соответствующей кнопки в окне с теоретическим материалом или окне с визуализацией;

- просмотр визуализации выбранного алгоритма – окно с визуализацией открывается после нажатия соответствующей кнопки в окне с теоретическим материалом или окне с блок-схемой;

- просмотр справки – можно посмотреть в верхнем меню вкладки «Справка»;

- возможность вернуться в главное меню для выбора нового метода.

2.1 Алгоритм Робертса

Алгоритм Робертса представляет собой первое известное решение задачи удалении невидимых линий. Это математически элегантный метод, работающий в объектном пространстве. Алгоритм прежде всего удаляет из каждого тела те ребра или грани, которые экранируются самим телом. Затем каждое из видимых

ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части, если таковые есть, экранируются этими телами. Поэтому вычислительная трудоемкость алгоритма Робертса растет теоретически как квадрат числа объектов. Однако математические методы, используемые в этом алгоритме, просты, мощны и точны.

Более поздние реализации алгоритма, использующие предварительную приоритетную сортировку вдоль оси z и простые габаритные или минимаксные тесты, демонстрируют почти линейную зависимость от числа объектов.

Работа Алгоритм Робертса проходит в два этапа:

1. Определение нелицевых граней для каждого тела отдельно.
2. Определение и удаление невидимых ребер.

Определение нелицевых граней

Пусть F — некоторая грань многогранника. Плоскость, несущая эту грань, разделяет пространство на два подпространства. Назовем положительным то из них, в которое смотрит внешняя нормаль к грани. Если точка наблюдения – в положительном подпространстве, то грань – **лицевая**, в противном случае – **нелицевая**. Если многогранник выпуклый, то удаление всех нелицевых граней полностью решает задачу визуализации с удалением невидимых граней.

Для определения, лежит ли точка в положительном подпространстве, используют проверку знака скалярного произведения (l, n), где l – вектор, направленный к наблюдателю, фактически определяет точку наблюдения; n – вектор внешней нормали грани. Если $(l, n) > 0$, т. е. угол между векторами острый, то грань является лицевой. Если $(l, n) < 0$, т. е. угол между векторами тупой, то грань является нелицевой.

Этот метод является простейшим алгоритмом удаления невидимых поверхностей для тел, представляющих собой одиночные выпуклые многогранники. Он также используется для удаления нелицевых или задних граней из сцены перед применением одного из алгоритмов удаления невидимых линий, которые обсуждаются ниже. Этот способ часто называют *отбрасыванием задних плоскостей*. Для выпуклых многогранников число граней при этом сокращается примерно наполовину. Метод эквивалентен вычислению нормали к поверхности для каждого отдельного многоугольника.

Данный метод определения нелицевых граней в результате формирует аксонометрическую проекцию на некую плоскость, расположенную бесконечно далеко от любой точки трехмерного пространства. Видовые преобразования,

включая перспективное, производятся до определения нелицевых плоскостей. Когда видовое преобразование включает в себя перспективу, то нужно использовать полное перспективное преобразование одного трехмерного пространства в другое, а не перспективное проецирование на некоторую двумерную плоскость. Полное перспективное преобразование приводит к искажению трехмерного тела, которое затем проецируется на некую плоскость в бесконечности, когда нелицевые плоскости уже определены. Этот результат эквивалентен перспективному проецированию из некоторого центра на конечную плоскость проекции.

Удаление невидимых ребер

После первого этапа удаления нелицевых отрезков необходимо выяснить, существуют ли такие отрезки, которые экранируются другими телами в картинке или в сцене. Для этого каждый оставшийся отрезок или ребро нужно сравнить с другими телами сцены или картинки.

Возможны следующие случаи:

- Грань ребра не закрывает. Ребро остается в списке ребер.
- Грань полностью закрывает ребро. Ребро удаляется из списка рассматриваемых ребер.
- 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.

Для оптимизации используется приоритетная сортировка (*z*-сортировка), а также сравнения с прямоугольными объемлющими оболочками тел. Такой подход позволяет удалить целые группы или кластеры отрезков и тел. Например, если все тела в сцене упорядочены в некотором приоритетном списке, использующем значения *z* ближайших вершин для представления расстояния до наблюдателя, то никакое тело из этого списка, у которого ближайшая вершина находится дальше от наблюдателя, чем самая удаленная из концевых точек ребра, не может закрывать это ребро. Более того, ни одно из оставшихся тел, прямоугольная оболочка которого расположена полностью справа, слева, над или под ребром, не может экранировать это ребро. Использование этих приемов значительно сокращает число тел, с которыми нужно сравнивать каждый отрезок или ребро. Рисунок 2.1 иллюстрирует работу алгоритма.

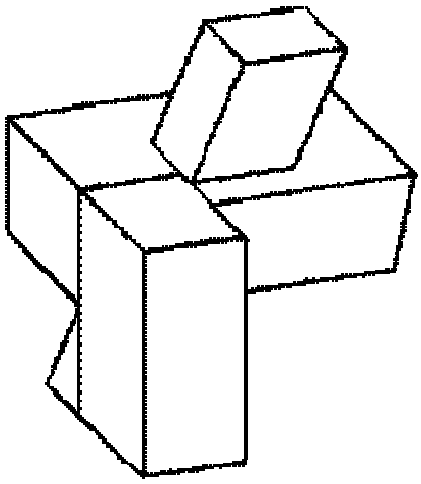


Рисунок 2.1 - Результат работы алгоритма Робертса

2.2 Алгоритм Аппеля

Алгоритм использует понятие количественной невидимости точки – количество лицевых граней, закрывающих данную точку. Очевидно, что видны будут только те точки, для которых количественная невидимость равна нулю. Для полигональных поверхностей количественная невидимость изменяется только на контурных линиях проекций этих поверхностей на экран. Это верно только для непересекающихся поверхностей.

Для определения видимости ребер некоторой поверхности сначала непосредственно определяется количественная невидимость одной из ее вершин, далее прослеживается изменение количественной невидимости вдоль ребер, выходящих из этой точки. Она изменяется на единицу при прохождении ребра за линией контура. Во входящих точках – увеличивается, в выходящих – уменьшается (для этого контур должен быть направленным). Части ребер с нулевой количественной невидимостью сразу рисуются. Операция повторяется для вершин на концах этих ребер (для них количественная невидимость уже определена), пока не будут проверены все вершины и ребра грани.

2.3 Метод сортировки по глубине

Наиболее простым подходом к упорядочиванию граней является их сортировка по минимальному расстоянию до картинной плоскости (вдоль направления проектирования) с последующим выводом их в порядке приближения.

Однако возможны случаи, когда просто сортировка по расстоянию до картинной плоскости не обеспечивает правильного упорядочения граней; поэтому желательно после такой сортировки проверить порядок, в котором грани будут выводиться.

Предлагается следующий алгоритм этой проверки. Для простоты считается, что рассматривается параллельное проектирование вдоль оси Oz.

Перед выводом грани P следует убедиться, что никакая другая грань Q, проекция которой на ось Oz пересекается с проекцией грани P, не может закрываться гранью P. И если это условие выполнено, то грань P должна быть выведена раньше.

Пять тестов:

1. x-Оболочки многоугольников не перекрываются, поэтому сами многоугольники тоже не перекрываются;

2. y-Оболочки многоугольников не перекрываются, поэтому сами многоугольники тоже не перекрываются;

3. P полностью расположен с той стороны от плоскости Q, которая дальше от точки зрения;

4. Q полностью расположен с той стороны от плоскости P, которая ближе к точке зрения;

5. Проекции многоугольников на плоскости xOy, то есть на экране, не перекрываются.

В случае если ни один из этих тестов не позволяет с уверенностью решить, какую из этих двух граней нужно выводить раньше, то одна из них разбивается плоскостью, проходящей через другую грань. В этом случае вопрос об упорядочении оставшейся грани и частей разбитой грани легко решается.

2.4 Метод Z буфера

Алгоритм, использующий *z-буфер* – один из простейших алгоритмов удаления невидимых поверхностей. Впервые он был предложен Кэтмулом. Работает этот алгоритм в пространстве изображения. Идея *z*-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пикселя в пространстве изображения, z-буфер – это отдельный буфер глубины, используемый для запоминания координаты *z* или глубины каждого видимого пикселя в пространстве изображения. В процессе работы 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 это сравнение показывает, что новый пиксель расположен впереди пикселя, находящегося в буфере кадра, то новый пиксель заносится в этот буфер и, кроме того, производится корректировка z-буфера новым значением z. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по x и y наибольшего значения функции $z\ (x, y)$.

Главное преимущество алгоритма – его простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей. Сцены могут быть любой сложности. Поскольку габариты пространства изображения фиксированы, оценка вычислительной трудоемкости алгоритма не более чем линейна. Поскольку элементы сцены или картинки можно заносить в буфер кадра или в z-буфер в произвольном порядке, их не нужно предварительно сортировать по приоритету глубины. Поэтому экономится вычислительное время, затрачиваемое на сортировку по глубине.

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. Информацию о глубине нужно обрабатывать с большей точностью, чем координатную информацию на плоскости (x, y); обычно бывает достаточно 20-ти бит. Буфер кадра размером 1024×768×24 бит в комбинации с z-буфером размером 1024×768×20 бит требует почти 1.5 мегабайт памяти. Однако снижение цен на память делает экономически оправданным создание специализированных запоминающих устройств для z-буфера и связанной с ним аппаратуры.

Формальное описание алгоритма z-буфера таково:

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 *Пиксель*(x,y) 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 атрибут этого многоугольника (интенсивность, цвет и т. п.) в буфер кадра и заменить Z *буфер*(x,y) на $z(x,y)$. В противном случае никаких действий не производить.

В качестве предварительного шага там, где это целесообразно, применяется удаление нелицевых граней.

Если известно уравнение плоскости, несущей каждый многоугольник, то вычисление глубины каждого пикселя на сканирующей строке можно проделать пошаговым способом. Грань при этом рисуется последовательно (строка за строкой). Для нахождения необходимых значений используется линейная интерполяция (рис. 2.2).

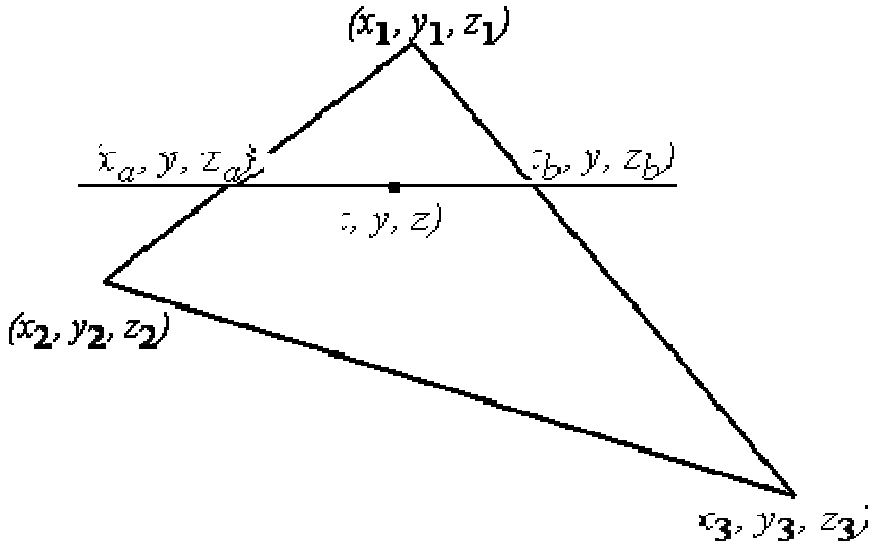


Рисунок 2.2 - Сканирующая строка по грани

Для рисунка y меняется от y_1 до y_2 и далее до y_3, при этом для каждой строки определяется x_a, z_a, x_b, z_b:

$$x_a = x_1 + (x_2 - x_1)\cdot \frac{y - y_1}{y_2 - y_1};$$

$$x_b = x_1 + (x_3 - x_1)\cdot \frac{y - y_1}{y_3 - y_1};$$

$$z_a = z_1 + (z_2 - z_1)\cdot \frac{y - y_1}{y_2 - y_1};$$

$$z_b = z_1 + (z_3 - z_1)\cdot \frac{y - y_1}{y_3 - y_1}.$$

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

$$z = z_a + (z_b - z_a)\cdot \frac{x - x_a}{x_b - x_a}$$

Реализация алгоритма вдоль сканирующей строки позволяет совместить метод z-буфера с алгоритмами растровой развертки ребер и алгоритмами закраски грани.

Иллюстрация работы алгоритма на примере для рис. 2.3.

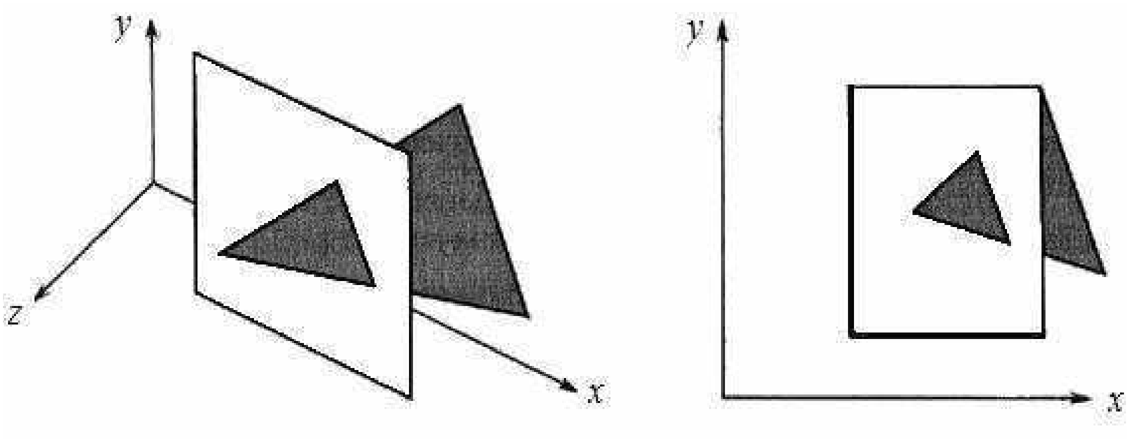


Рисунок 2.3 - Протыкающий треугольник

В начале в буфере кадра и в z-буфере содержатся нули. После растровой развертки прямоугольника содержимое буфера кадра будет иметь вид

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Содержимое *z*-буфера таково:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

При обработке треугольника преобразование его в растровую форму и сравнение по глубине дает новое значение буфера кадра:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	2	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	2	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	2	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	2	0	0
0	0	0	1	1	1	1	1	1	1	1	2	1	1	1	1	2	2	0
0	0	0	1	1	1	1	1	1	1	2	2	1	1	1	1	2	2	0
0	0	0	1	1	1	1	1	1	2	2	2	2	1	1	1	2	2	0
0	0	0	1	1	1	1	1	2	2	2	2	2	1	1	1	2	2	0
0	0	0	1	1	1	1	2	2	2	2	2	2	1	1	1	2	2	0
0	0	0	1	1	1	1	1	2	2	2	2	2	2	1	1	2	2	2
0	0	0	1	1	1	1	1	1	1	1	2	2	2	1	1	2	2	2
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Новое содержимое *z*-буфера таково:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	5	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	5	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	6	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	6	0	0
0	0	0	10	10	10	10	10	10	10	10	11	10	10	10	10	6	5	0
0	0	0	10	10	10	10	10	10	10	12	11	10	10	10	10	7	5	0
0	0	0	10	10	10	10	10	10	13	12	11	11	10	10	10	7	6	0
0	0	0	10	10	10	10	10	14	13	12	12	11	10	10	10	7	6	0
0	0	0	10	10	10	10	15	14	13	12	12	11	10	10	10	7	6	0
0	0	0	10	10	10	10	10	14	13	12	12	11	11	10	10	8	7	5
0	0	0	10	10	10	10	10	10	10	10	12	11	11	10	10	8	7	5
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	8	7	6
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	6
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

2.5 Метод построчного сканирования

Метод построчного сканирования является еще одним примером метода, работающего в пространстве картинной плоскости. Однако вместо того, чтобы решать задачу удаления невидимых граней для проекций объектов на картинную плоскость, задача приводится к серии простых одномерных задач. Есть возможность представить все изображение на картинной плоскости, как ряд горизонтальных (вертикальных) линий пикселей. Проводится анализ сечение сцены плоскостью, проходящей через такую линию пикселей и центр проектирования. Пересечением этой плоскости с объектами сцены будет множество непересекающихся (за исключением концов) отрезков, которые и необходимо спроектировать. Задача удаления невидимых частей для такого набора отрезков решается тривиально. Рассматривая задачу удаления невидимых граней для каждой такой линии, тем самым происходит разбитие исходной задачи на набор гораздо более простых задач.

3 ПЛАНИРОВАНИЕ

3.1 Платформа

Для реализации демонстрационного комплекса программ выбрана платформа Windows 7 32 bit, исходя из требований заказчика. В настоящий момент Windows является самой распространенной операционной системой [18], к тому же в большинстве учебных заведений в Российской Федерации, на компьютерах установлена именно она. Программы, разработанные на выбранной 32-х битной архитектуре Windows, способны работать как на 32-х битной системе Windows, так и на 64-х битной, в то время, как программы написанные под 64-х битную систему, не способны работать в среде 32-х разрядной системы.

3.2 Возможные варианты реализации

Возможные средства реализации демонстрационного комплекса программ по большей части представляют собой совокупность различных языков программирования с различными графическими API. Так как демонстрационный комплекс программ – это приложение для Windows, то из языков программирования на данный момент самые распространенные для подобных проектов это C, C++, C#, Java. Для работы с графикой в приоритете такие графические API, как OpenGL, DirectX [19], Vulkan [19], также рассмотрению подлежали и технологии для работы с графикой .NET Framework[2]. В зависимости от выбранного языка, программирование может быть структурным либо объектно-ориентированным. Может использоваться технология многопоточного программирования.

3.3 Обоснование среды разработки

Наиболее рационально использовать среду разработки Visual Studio 2017 (VS). Это бесплатный продукт компании Microsoft, который предоставляет мощный и удобный инструментарий для разработки. Также его достоинством является то, что среда разработки регулярно обновляется, что упрощает написание программ с каждым годом всё больше. С ее помощью можно разрабатывать помимо консольных приложений приложения с графическим интерфейсом, что необходимо для демонстрационного комплекса программ. VS не привязана к определенному языку программирования, что позволяет выбрать необходимый язык при создании проекта. Редактор форм VS упрощает создание графического интерфейса. В VS есть 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.

3.4 Выбор и обоснование средств разработки

Разрабатываемый продукт должен являться desktop-приложением (запускается в окне на рабочем столе) для Windows 7 SP1 x32, поэтому подходящим является

язык программирования C#. Если бы требовалась кроссплатформенность приложения, то выбор был бы между C++ и Java. Преимущество C# перед C – объектно-ориентированность, что удобнее для разработки объемных программ. Также C# ориентирован на безопасность кода по сравнению с C и C++. 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 демонстрационном комплексе программ не требуется очень быстрых вычислений, то недостаток не является значимым.

Выбранный интерфейс программирования приложения – Windows Form. Он упрощает доступ к элементам интерфейса Windows за счет создания обёртки для существующего Win32 API в управляемом коде, отвечает за графический интерфейс пользователя. Windows Form входит в .NET Framework, поэтому на компьютерах с операционной системой более ранней, чем Windows 7 SP1, нужно перед использованием демонстрационного комплекса программ устанавливать .NET Framework 4.7.

Для работы с графикой выбрана внутренняя библиотека .NET Framework Drawing. Выбор основан на том, что:

- работа алгоритмов в демонстрационном комплексе программ идет лишь с графическими примитивами, не требующими современных эффектов,
- код, написанный с использованием Drawing, выглядит нагляднее, чем с использованием DirectX или Vulcan, что проще для восприятия пользователям демонстрационном комплексе программ.

3.6 Архитектура системы

UML – это унифицированный язык моделирования, позволяющий строить модели программных систем. Он не зависит от выбранной методологии разработки ПО [13]. Исходя из вида диаграммы UML, модель системы может описывать функционал системы и ее взаимодействие с пользователями, архитектуру приложения; разработчик может писать каркасный код проектируемого приложения, используя диаграмму классов UML. Также UML облегчает обмен информацией в проектной группе и между заказчиком и исполнителем.

Для отражения предоставляемого функционала системы для разных групп пользователей полезно построить диаграмму прецедентов (диаграмму вариантов использования). Основные группы пользователей разрабатываемого проекта:

- преподаватели компьютерной графики - могут использовать демонстрационный комплекс программ для проведения лекций/практических занятий по темам растровых алгоритмов;
- студенты, которые изучают на компьютерной графике растровые алгоритмы.

В демонстрационном комплексе программ (ДКП) функционал для перечисленных выше групп пользователей одинаков.

Диаграммы прецедентов либо диаграммы вариантов использования отражают предоставляемый функционал системы. На диаграмме использования (см. рисунок 3.1) отражены сценарии работы с системой, которые приводят к желаемому пользователем результату.

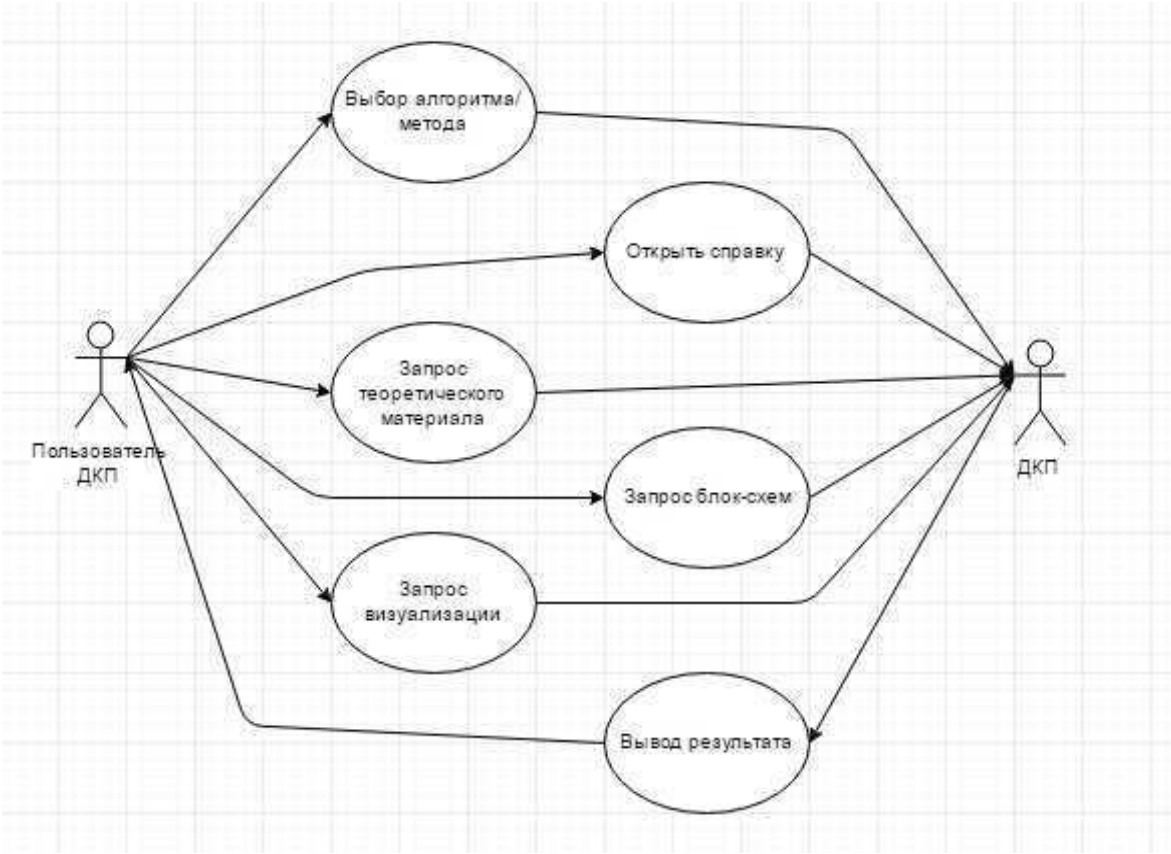


Рисунок 3.1 - Диаграмма использования

Объектная модель системы

Диаграмма классов — диаграмма, демонстрирующая классы системы, их атрибуты, методы и взаимосвязи между ними. На основе диаграммы использования (см. рисунок 3.2) составлена диаграмма классов для проекта (см. рисунок 3.1).

Рисунок 3.2 - Диаграмма классов

Архитектура

Для функционирования системы «Демонстрационный комплекс программ для изучения алгоритмов загораживания» в состав программы входят следующие внешние библиотеки: «Drawing» – для построения фигур.

Диаграмма компонентов представлена на рисунке 3.3.

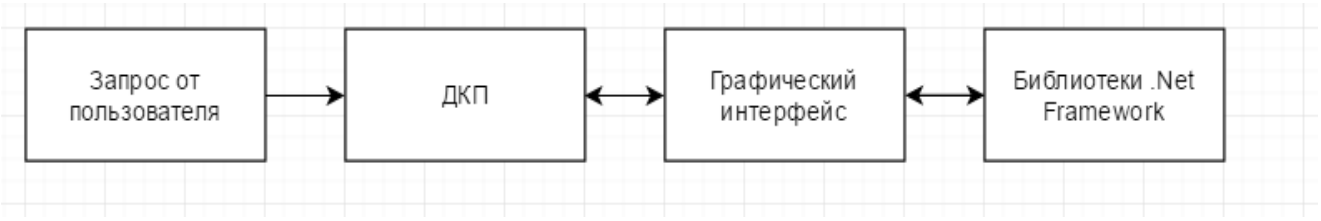


Рисунок 3.3 - Диаграмма компонентов

4 Основные алгоритмические решения

Для демонстрационного комплекса программ для изучения алгоритмов загораживания были подготовлены блок-схемы по каждому алгоритму загораживания, их теоретическая часть (см. раздел 2) и визуализация методов и алгоритмов.

4.1 Блок-схемы к методам и алгоритмам

В работе будут представлены укрупнённые схемы алгоритмов.

Ещё до начала работы самих алгоритмов, происходит удаление граней экранируемых самим объектом.

На рисунке 4.1 представлена укрупнённая блок-схема удаления невидимых граней. Для удаления используется тест DT2 [21].

Рисунок 4.1 - Блок-схема удаления невидимых граней

4.1.1 Алгоритм Робертса

На рисунке 4.2 представлена укрупнённая блок-схема алгоритма Робертса

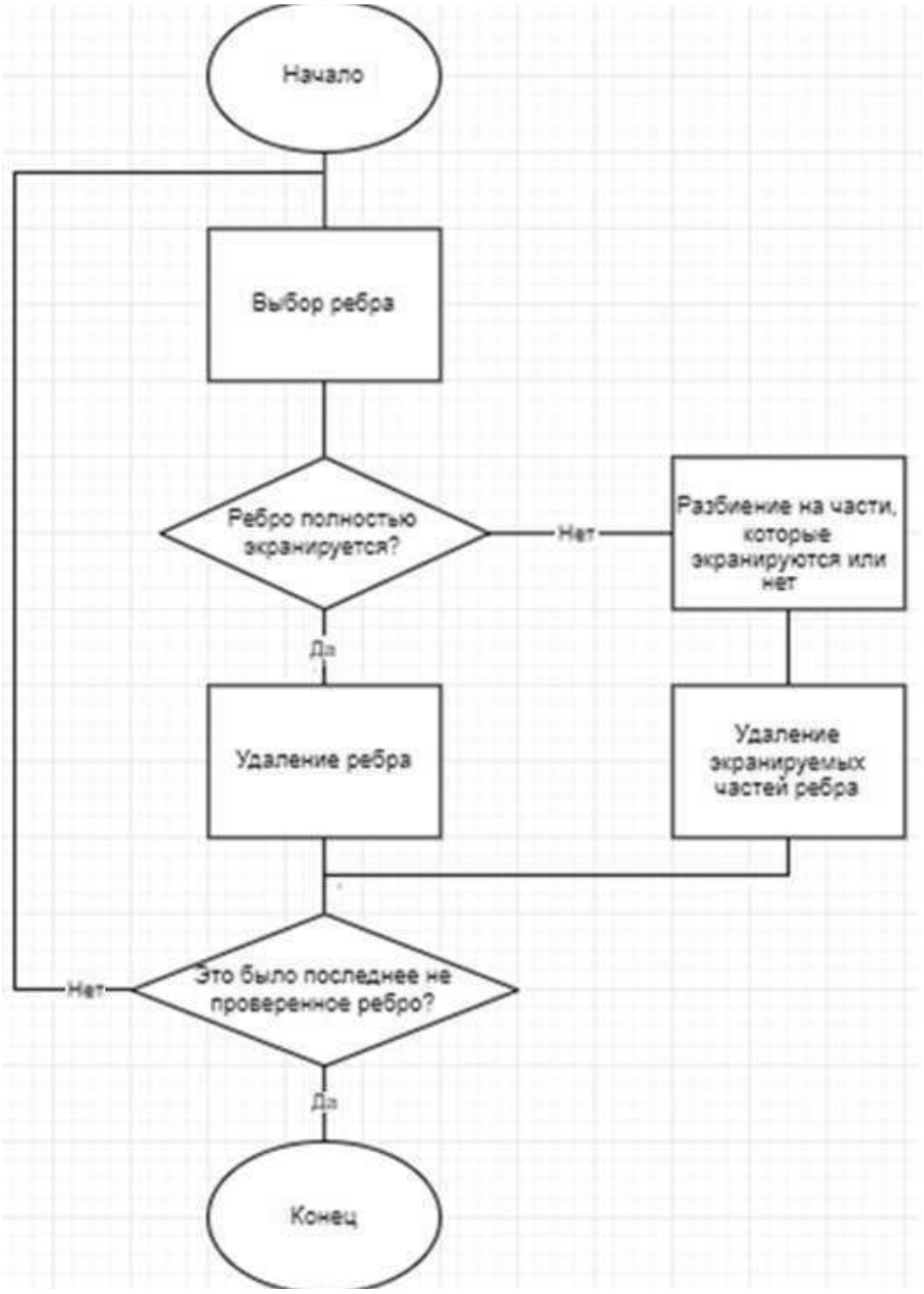


Рисунок 4.2 - Блок-схема алгоритма Робертса

4.1.2 Алгоритм Аппеля

На рисунке 4.3 представлена укрупнённая блок-схема алгоритма Аппеля

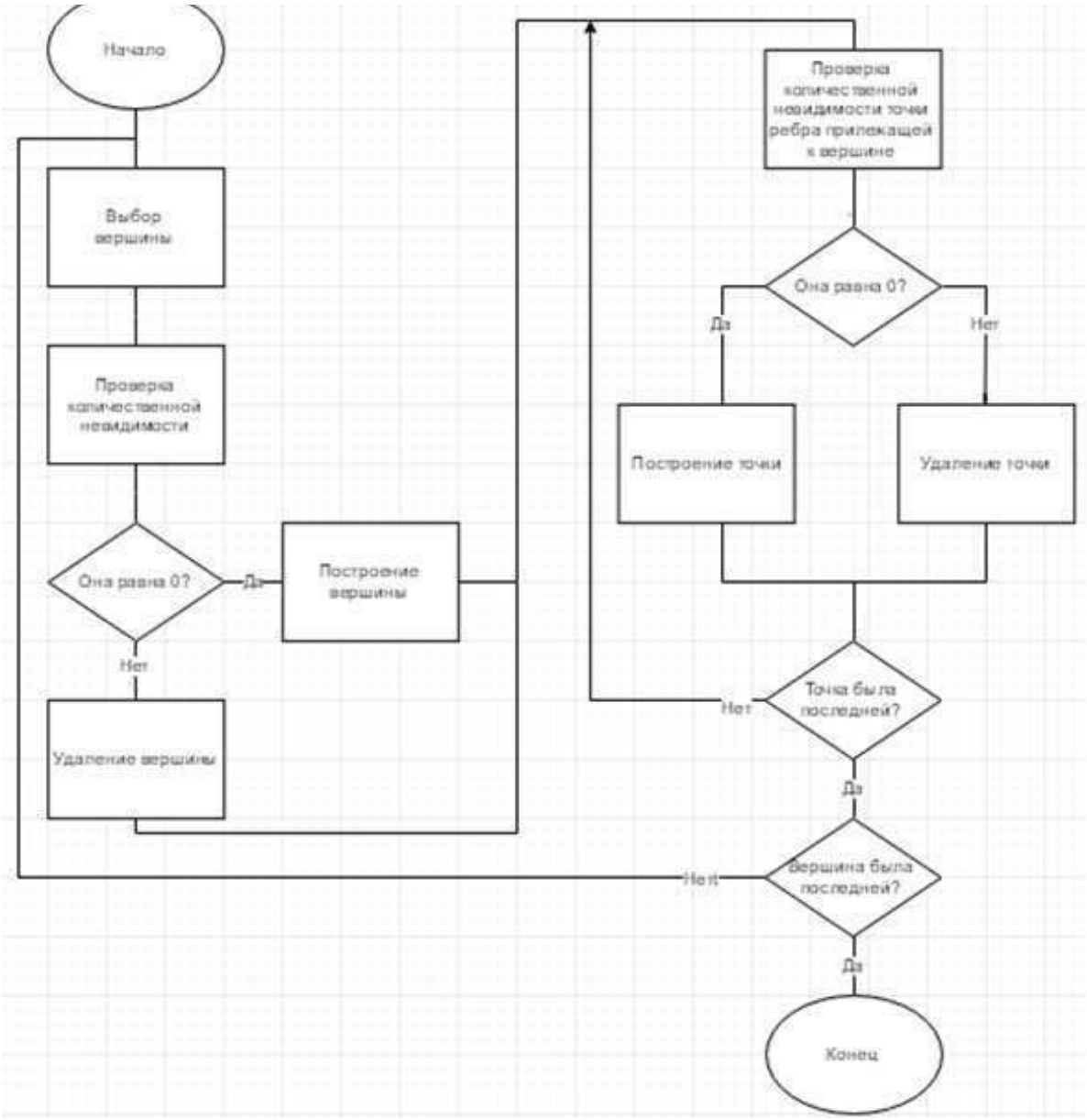


Рисунок 4.3 - Блок-схема алгоритма Аппеля

4.1.3 Метод сортировки по глубине

На рисунке 4.4 представлена укрупнённая блок-схема метода сортировки по глубине.

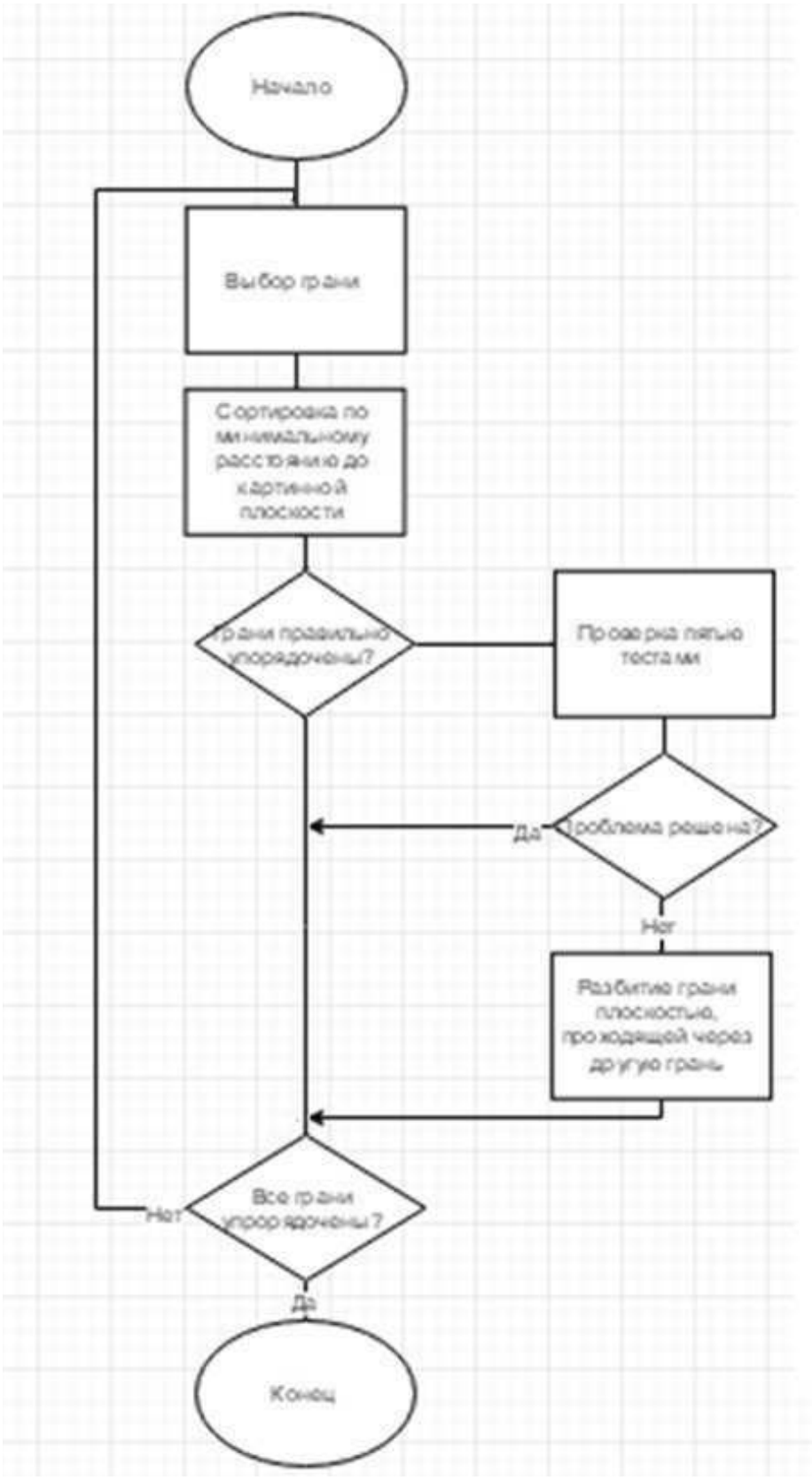


Рисунок 4.4 - Блок-схема метода сортировки по глубине

4.1.4 Метод Z-буфера

На рисунке 4.5 представлена укрупнённая блок-схема метода Z-буфера

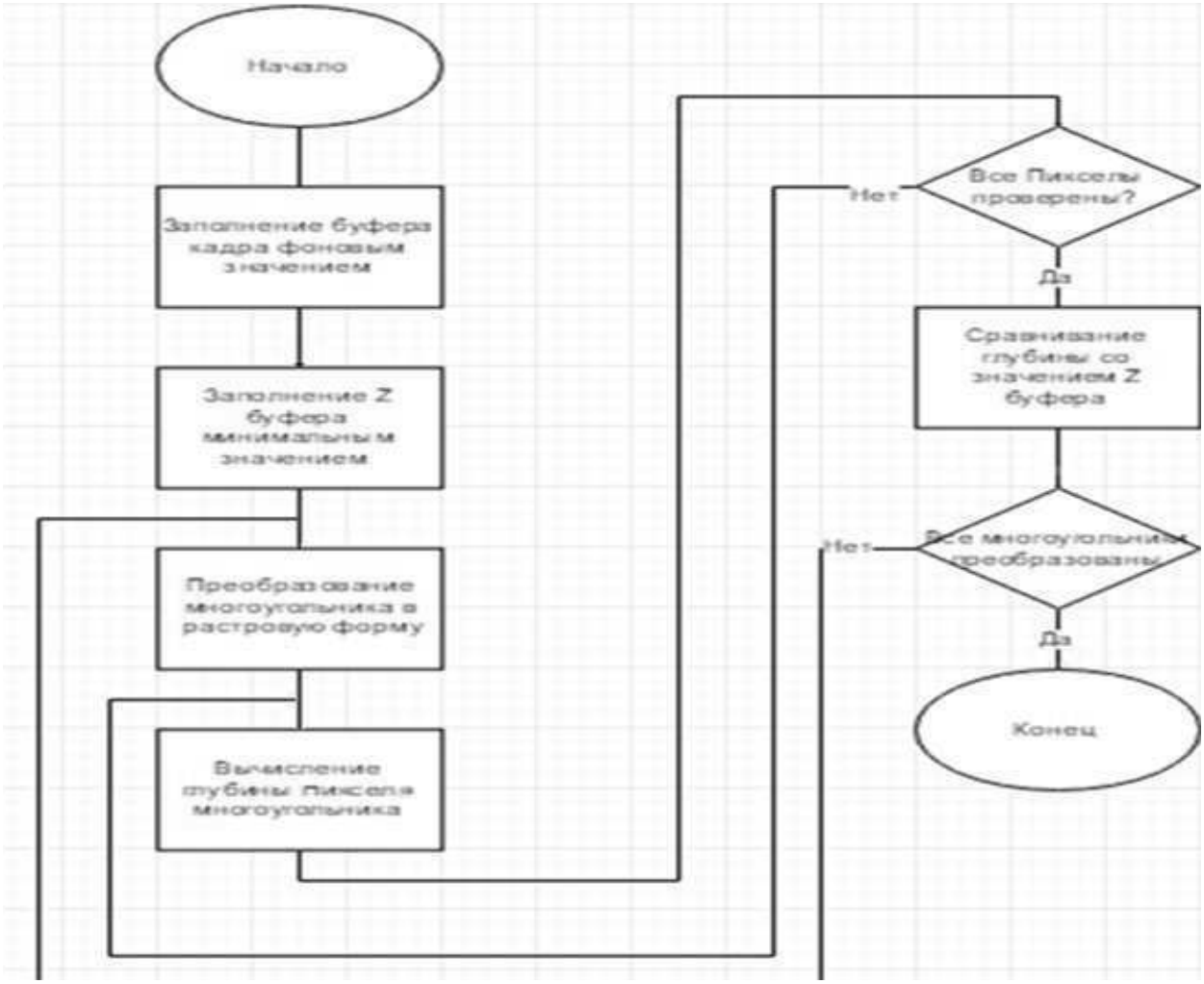


Рисунок 4.5 - Блок-схема метода Z-буфера

4.1.5 Метод построчного сканирования

На рисунке 4.6 представлена укрупнённая блок-схема метода построчного сканирования.

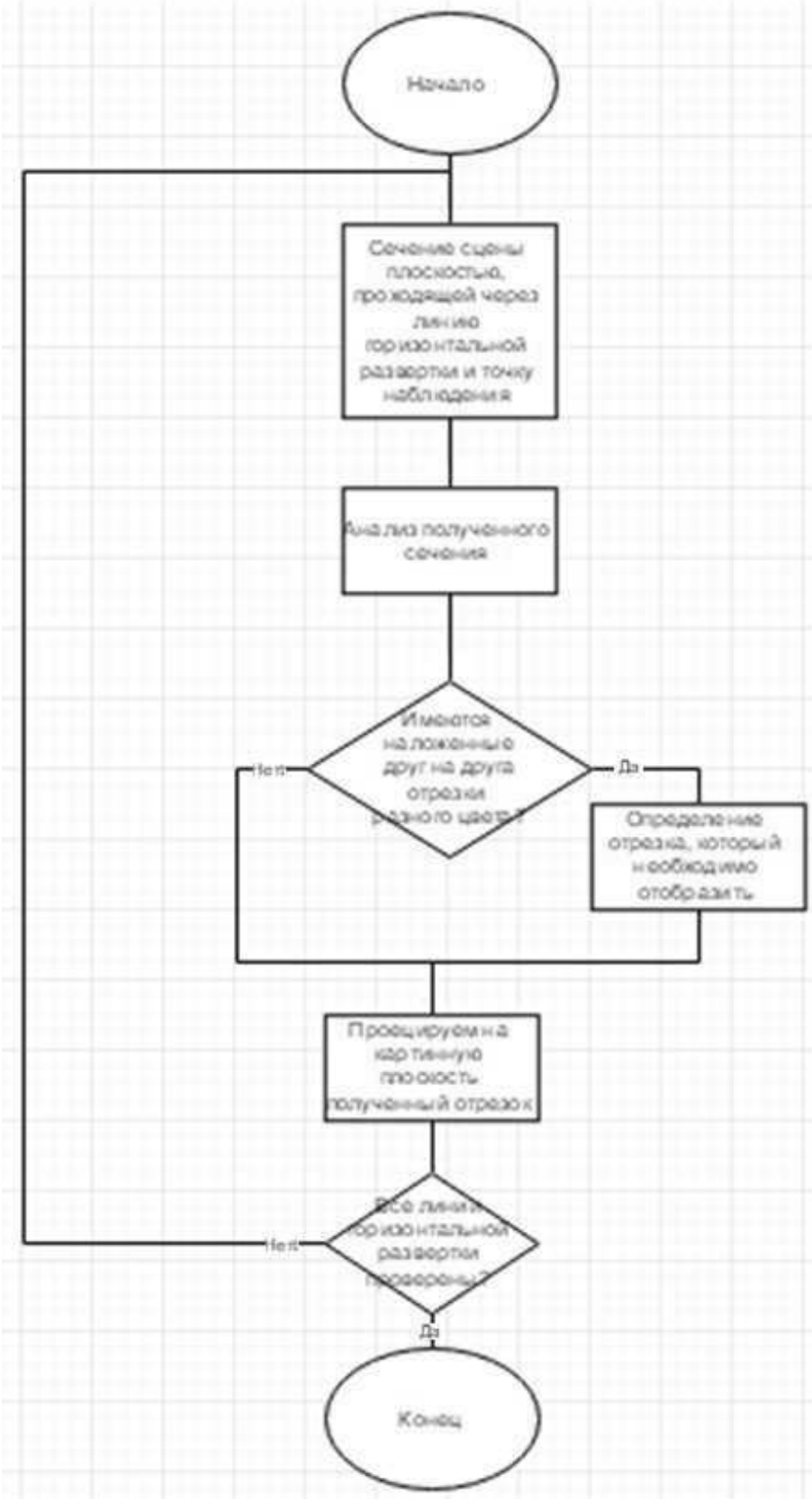


Рисунок 4.5 - Блок-схема метода построчного сканирования

4.2. Визуализация.

Начальную часть визуализации составляет построение объекта исследования (два пересекающихся прямоугольника). Для этого используется функция рисования линий из библиотеки Drawing DrawLine(Pen, Point, Point), где Pen – инструмент рисования, который также необходимо инициализировать Pen(Color, Single), Color – цвет карандаша, а Single – ширина. Первая Point отвечает за точку начала отрезка, вторая Point отвечает за точку конца отрезка. Но для начала построения линий необходимо инициализировать пространство, в котором будут проводиться графические преобразования.

Функция Bitmap(int,int); инициализирует новый экземпляр Bitmap класса с указанным размером. А Graphics.FromImage(image); позволяет работать с пространством Image. Пространство изображения pictureBox7 имеет размеры 1150 в высоту и 550 в ширину.

```
myBitmap = new Bitmap(pictureBox7.Width, pictureBox7.Height);
pictureBox7.Image = myBitmap;
g = Graphics.FromImage(myBitmap);
```

Далее проводится построение видимой части двух прямоугольников (см. приложение А).

За ней строятся невидимые части прямоугольников (см. приложение Б).

Рисуются координатные прямые.

```
g.DrawLine(myWind, new Point(325, 285), new Point(325, 10));
g.DrawLine(myWind, new Point(310, 275), new Point(1090, 275));
g.DrawLine(myWind, new Point(340, 265), new Point(-150, 550));
```

После построения изображения включается таймер, в котором в независимости от выбранного метода происходит исключение граней, которые полностью экранируются самим телом (см. приложение В).

В зависимости от выбранного метода далее происходят различия в зависимости от выбранного алгоритма:

код работы метода построчного сканирования представлен в приложении Г;

код выполнения работы алгоритм Аппеля представлен в приложении Д;

5 РЕЗУЛЬТАТЫ РЕАЛИЗАЦИИ

На рисунке 5.1 представлено главное меню.

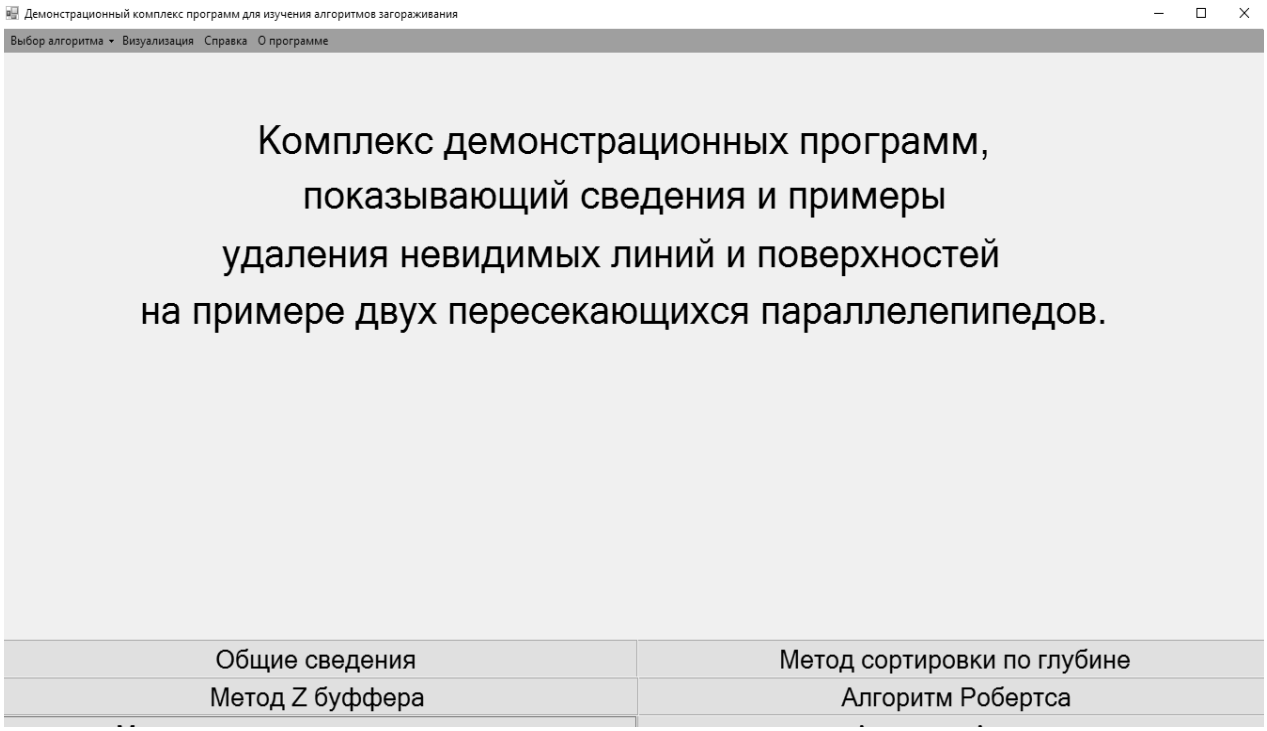


Рисунок 5.1 - Главное меню

На рисунках 5.2 и 5.3 представлены части теоретического материала по алгоритмам. На рисунке 5.2 представлено определение нелицевых граней в алгоритме Робертса, на 5.3 – начало описания метода Z буфера

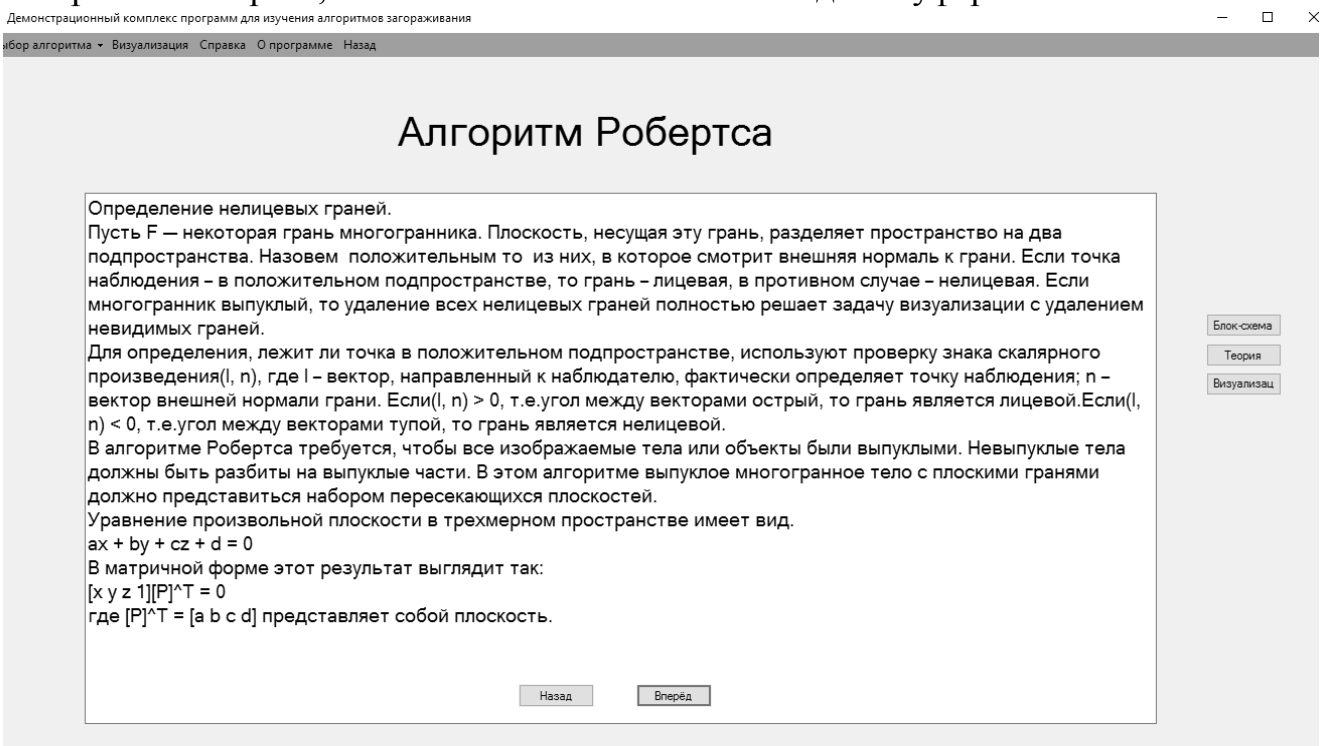


Рисунок 5.2 - Определение нелицевых граней в алгоритме Робертса

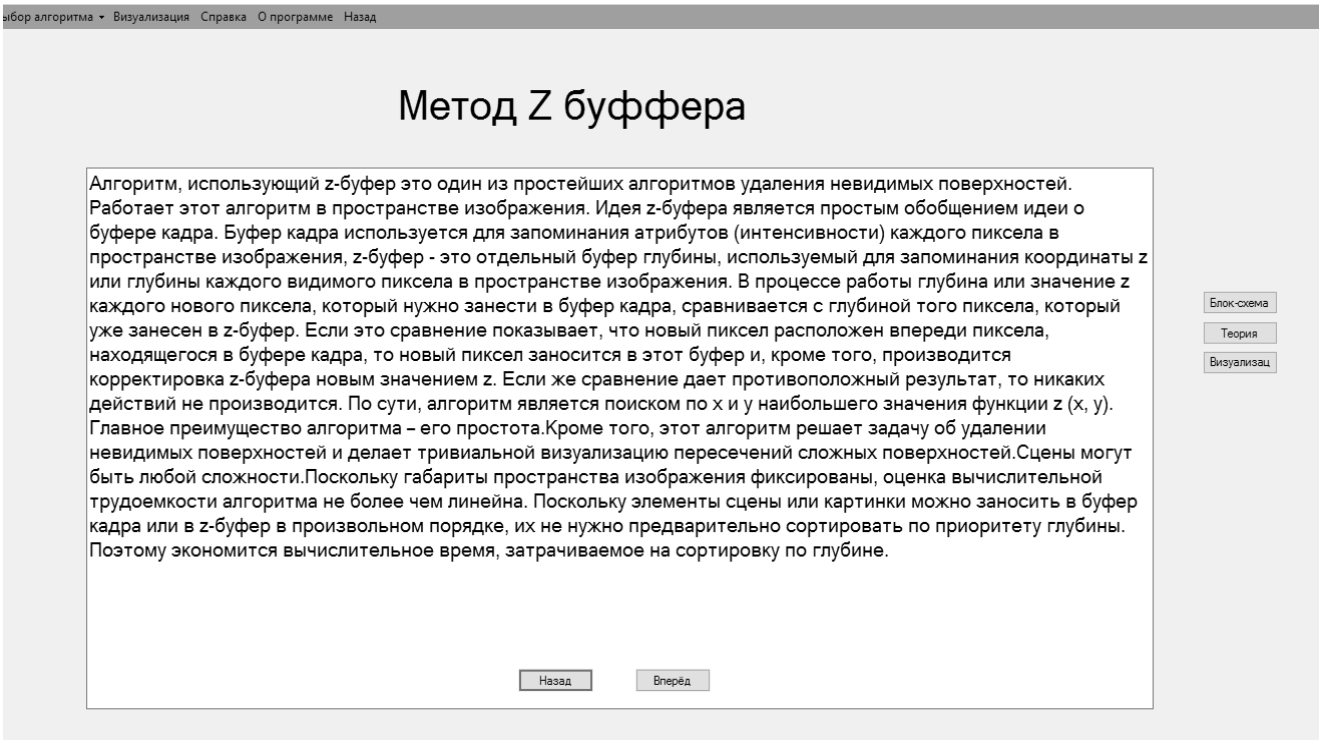


Рисунок 5.3 - Описание Z буфера

На рисунках 5.4 и 5.5 представлены блок-схемы. На рисунке 5.4 блок-схема метода Z буфера, а на 5.5 представлена блок-схема алгоритма Аппеля.

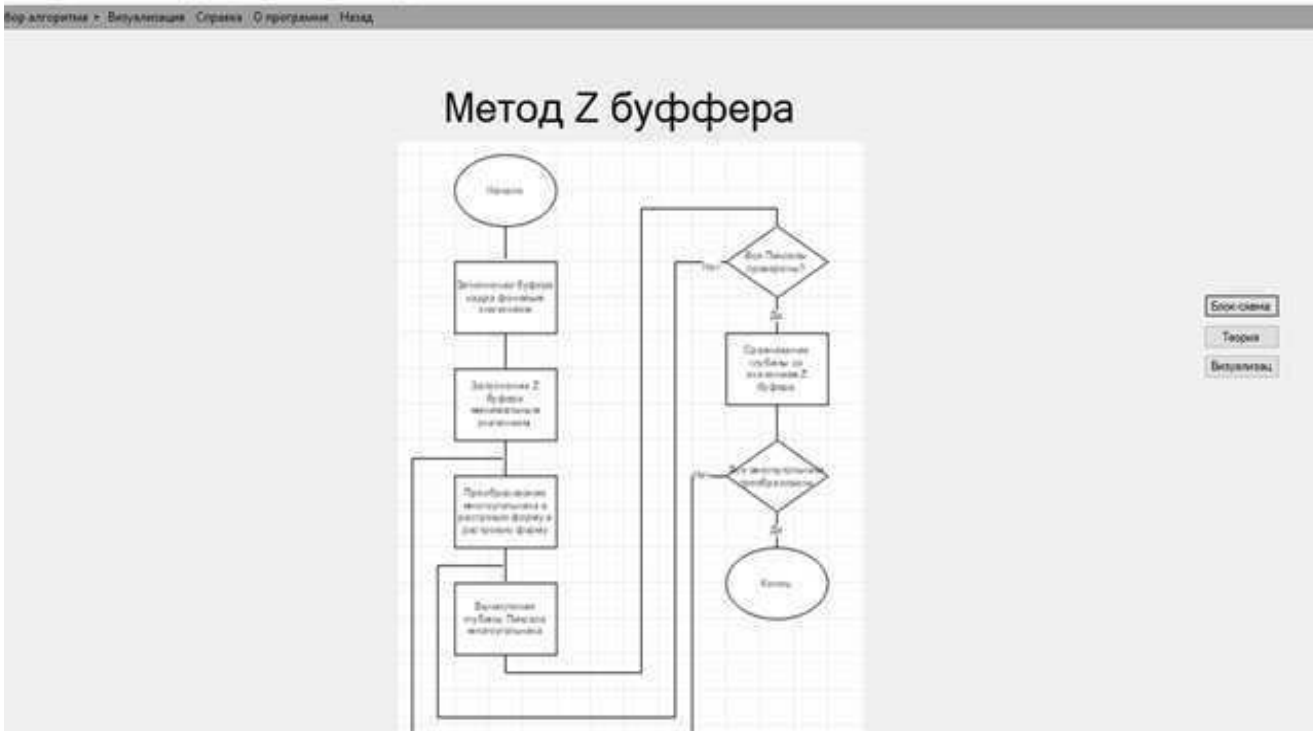


Рисунок 5.4 - Блок-схема метода Z буфера

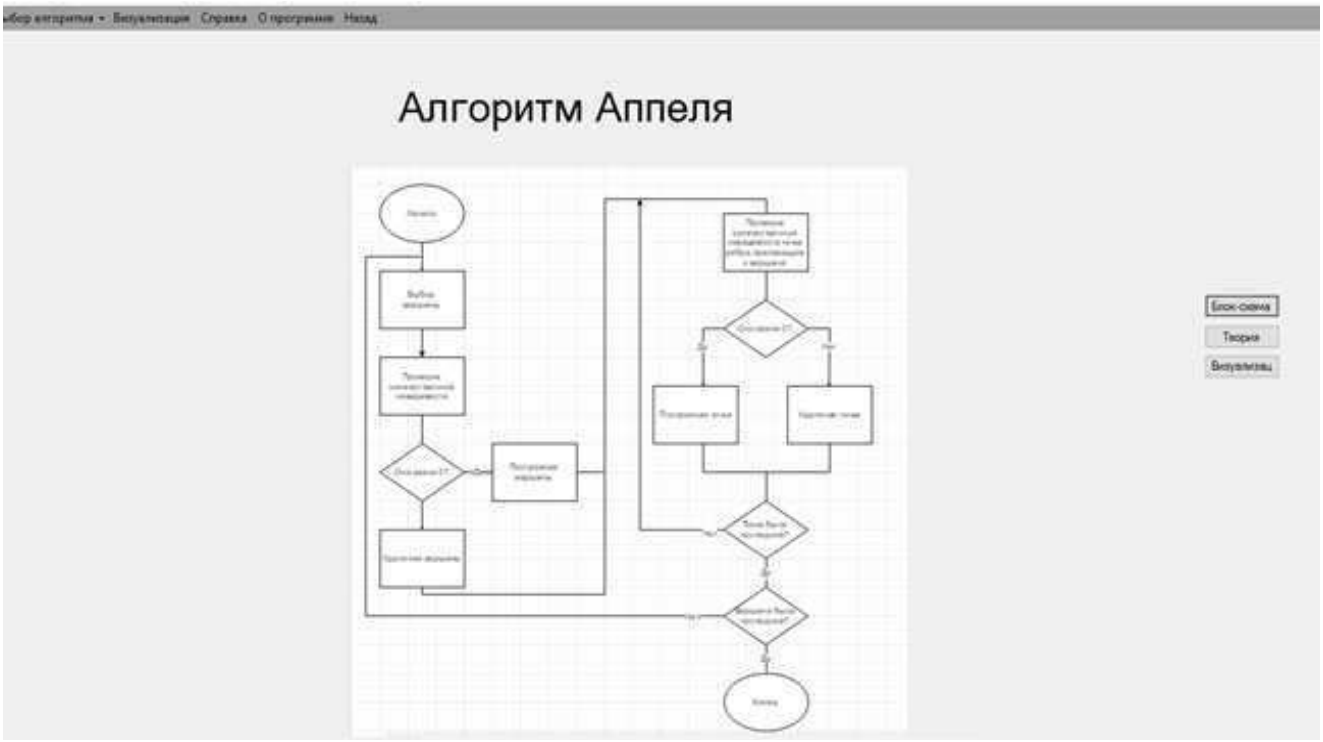


Рисунок 5.5 - Блок-схема алгоритма Аппеля

На рисунках 5.6-5.8 демонстрируется состояния. На рисунке 5.6 представлено начальное состояние объекта, а на 5.7 - состояние после прохождения алгоритма, на рисунке 5.8 состояние с которого начинается работа алгоритма

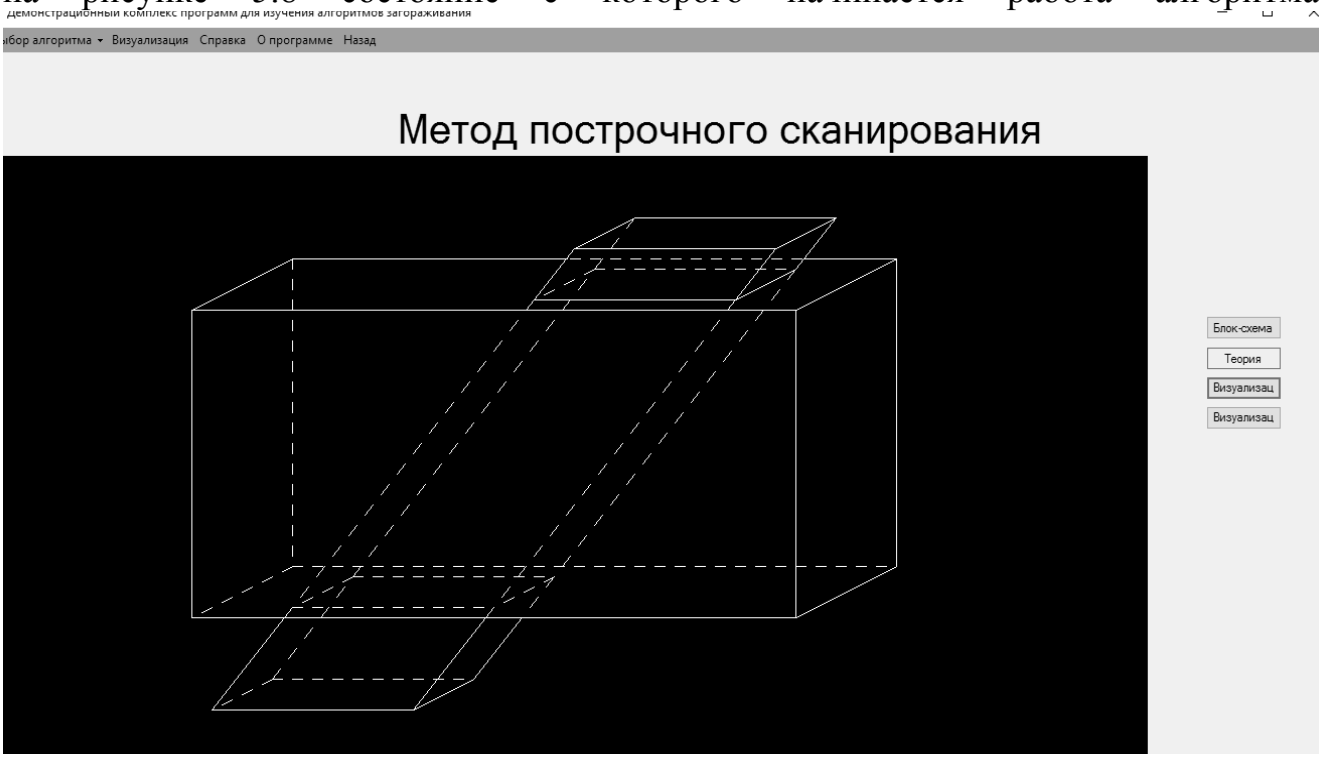


Рисунок 5.6 - Начальное состояние

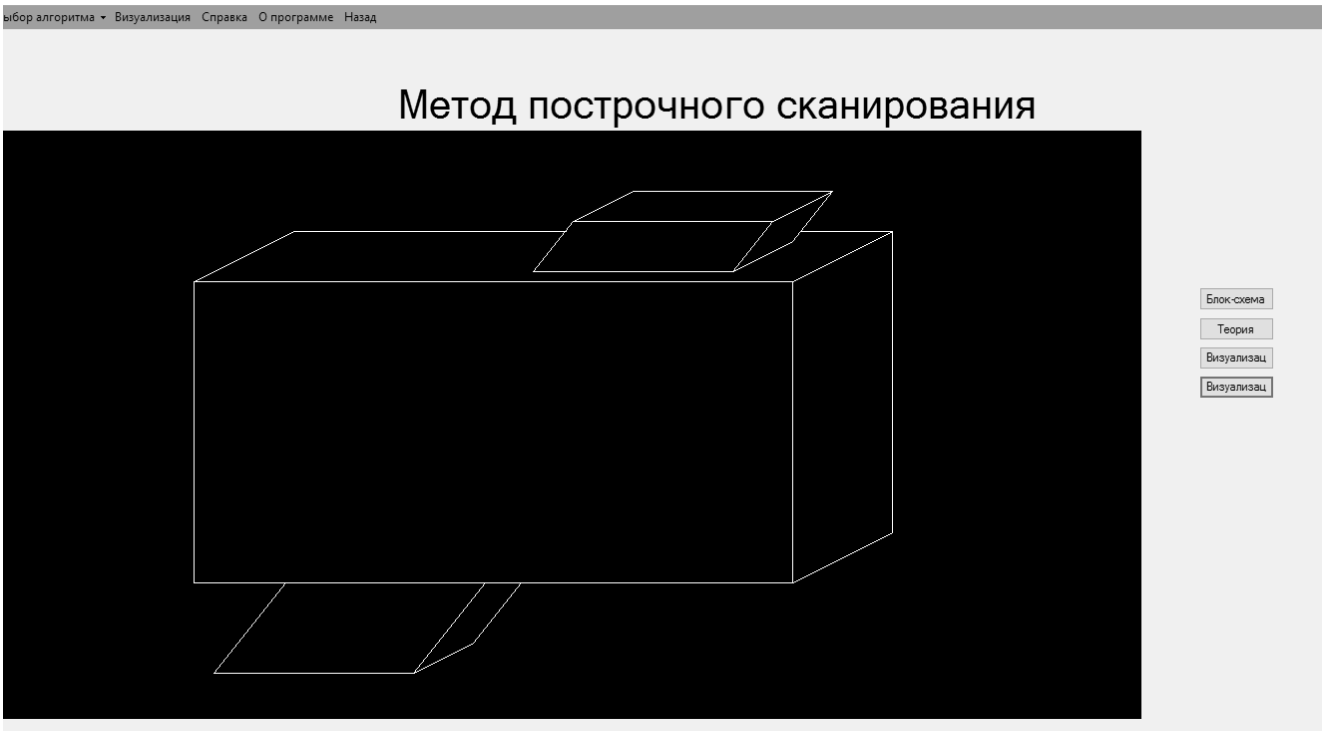


Рисунок 5.7 - Состояние после прохождения алгоритма

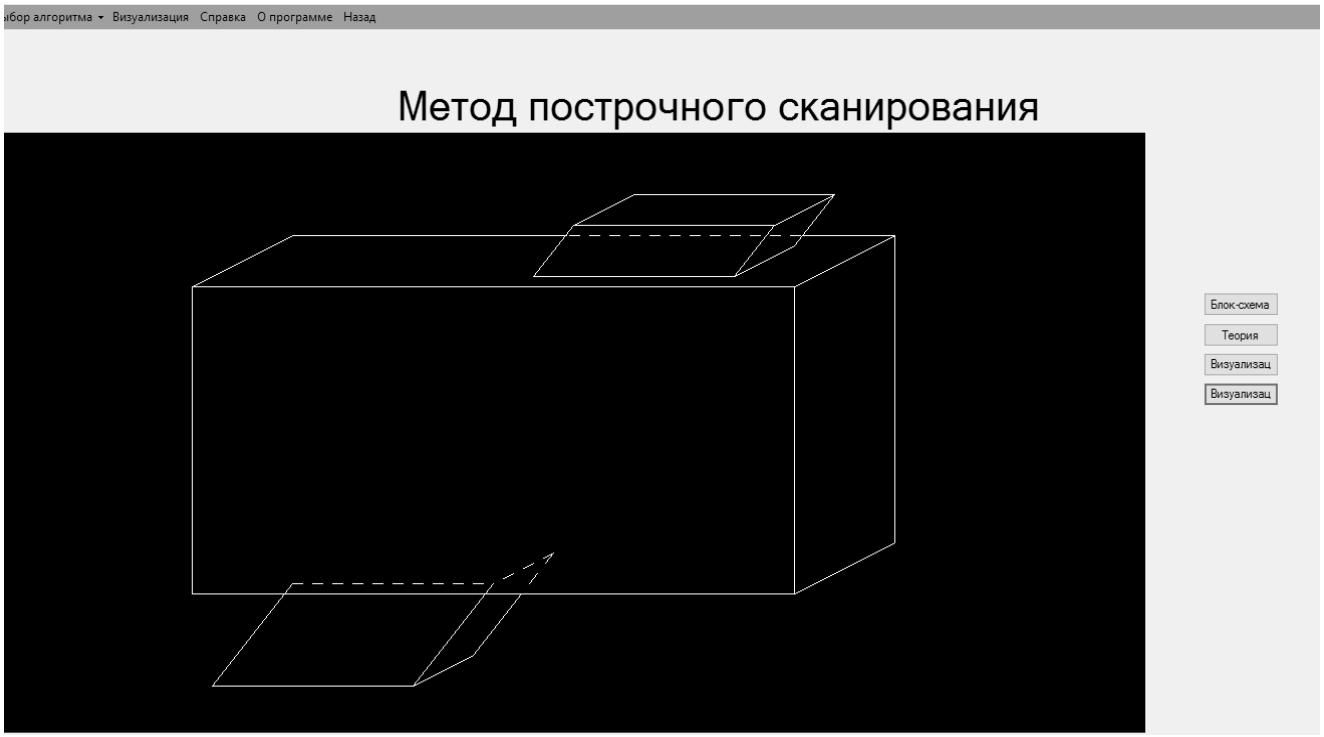


Рисунок 5.8 - Состояние с которого начинается ход алгоритма

На рисунке 5.9 представлено ход удаления экранируемых граней. На рисунках 5.10-5.13 представлен ход выполнения методов и алгоритмов удаления невидимых линий и поверхностей. На рисунке 5.10 ход выполнения метода построчного сканирования. На рисунке 5.11 ход выполнения алгоритма Аппеля. На рисунке 5.12 ход выполнения метод Z буфера. На рисунке 5.13 ход выполнения алгоритма Робертса.

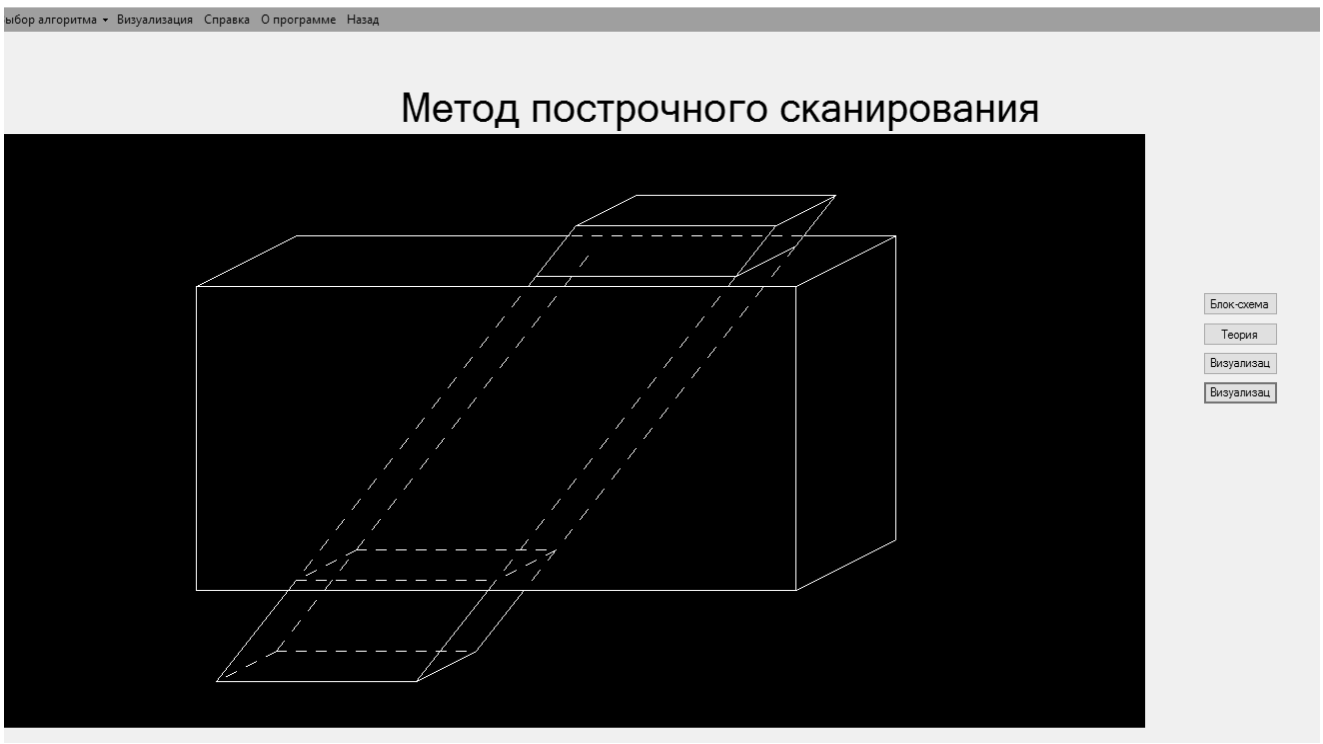


Рисунок 5.9 - Ход удаления экранируемых граней

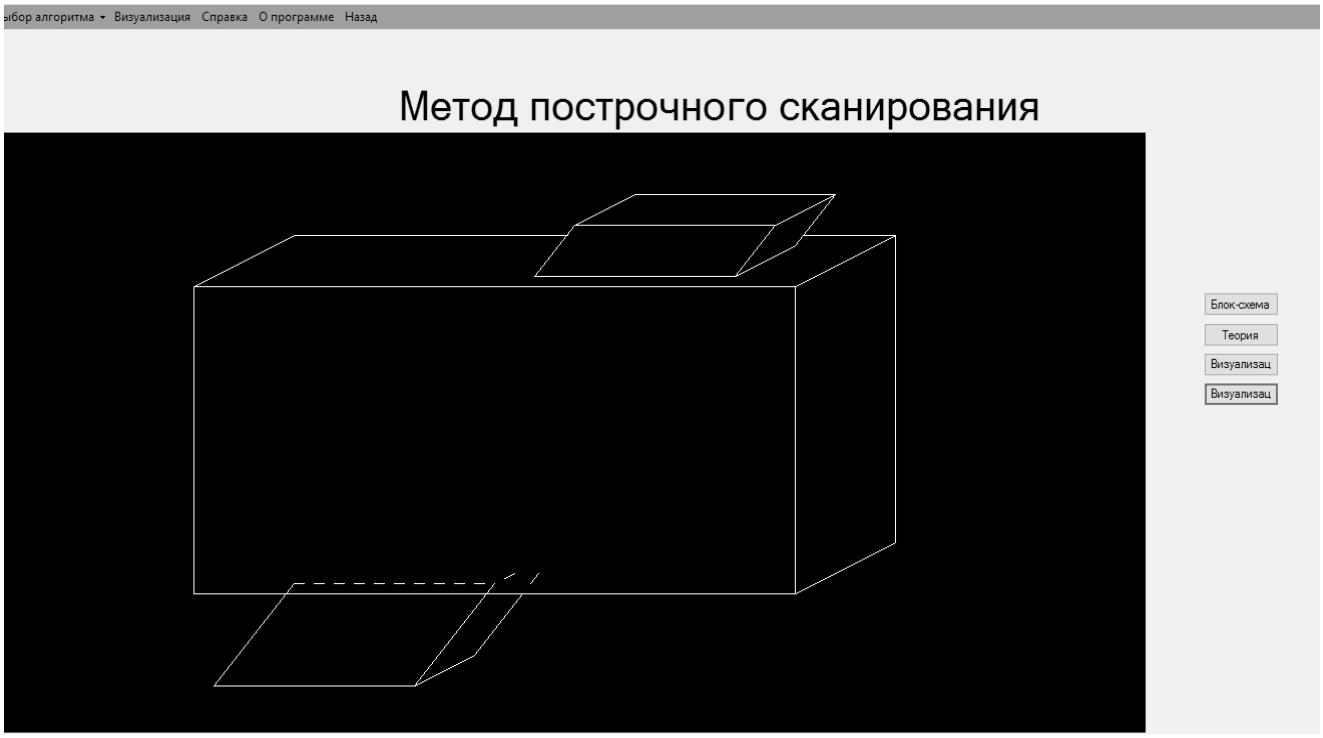


Рисунок 5.10 - Ход выполнения метода построчного сканирования

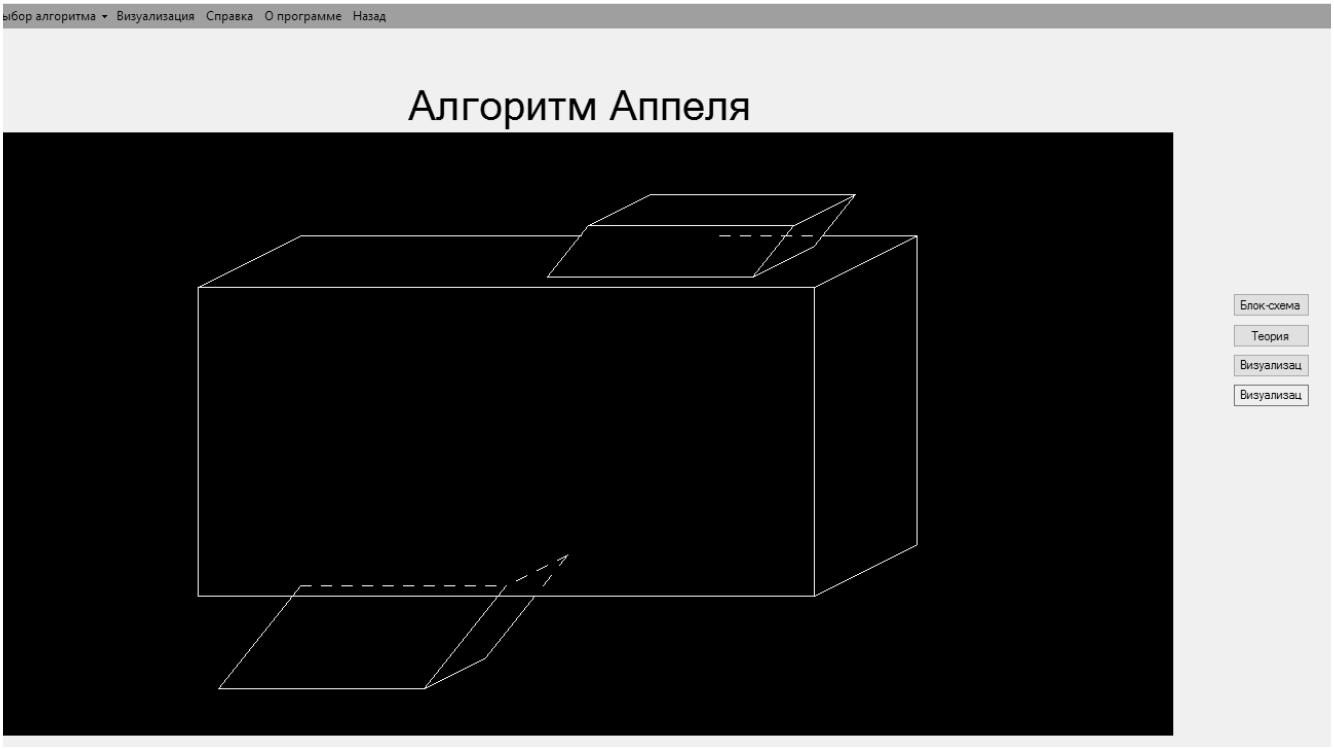


Рисунок 5.11 - Ход выполнения алгоритм Аппеля

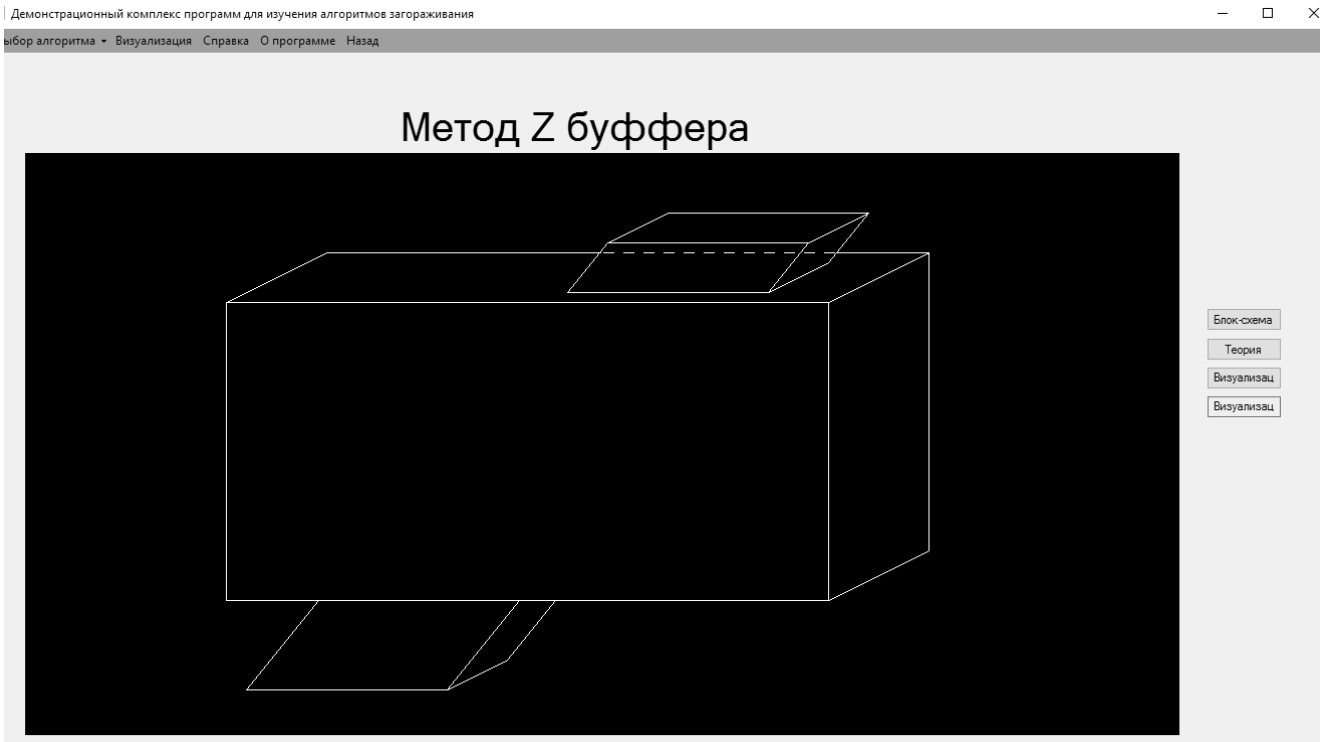


Рисунок 5.12 - Ход выполнения метода Z буфера

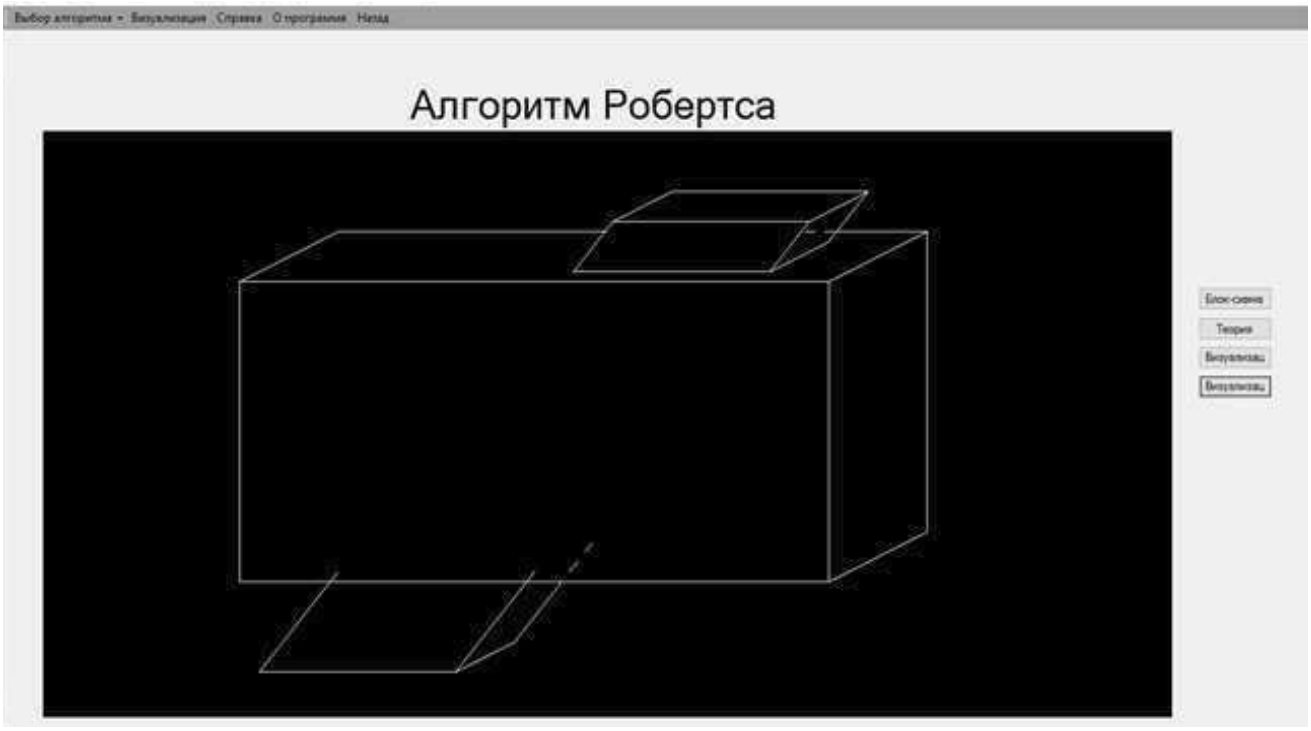


Рисунок 5.13 - Ход выполнения Алгоритм Робертса

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было разработано оконное приложение - «Демонстрационный комплекс программ для изучения алгоритмов загораживания» для Windows 7 x32 SP1.

В ходе выполнения работы решены такие задачи, как:

1. обоснована актуальность разработки,
2. проведен обзор аналогов приложения;
3. выбраны инструменты разработки;
4. спроектирована программная система;
5. реализован демонстрационный комплекс программ.

В приложение были включены наиболее распространенные алгоритмы удаления невидимых линий и поверхностей:

- алгоритм Робертса;
- алгоритм Аппеля;
- метод сортировки по глубине;
- метод Z-буфера;
- метод построчного сканирования;

А также исправлены такие недостатки найденных аналогов, как:

- отсутствие пояснений,
- отсутствие наглядности построения,
- недостоверность алгоритмов.

На основании учебно-методического пособия Ярош Е.С «Методы и средства представления графической информации» разработаны исходные коды перечисленных выше алгоритмов. Для каждого алгоритма была реализована анимация построения с выводом выполняемого кода за счёт многопоточного программирования.

Кроме модуля визуализации разработаны модули:

- теоретический материал;
- графический материал - блок-схемы.

Модули используют данные из Ярош Е.С «Методы и средства представления графической информации».

Организована навигация по всему приложению и создан эргономичный интерфейс.

Демонстрационный комплекс программ написан на языке программирования C#. Для работы с графикой использовались внешние библиотеки – Drawing (.Net Framework). Разработка приложения закончена и передана заказчику.

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

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Компьютерная графика. Уроки, алгоритмы, программы, примеры [Электронный ресурс]. – Режим доступа: http://www.grafika.me/lessons. – Заглавие с экрана. – (Дата обращения: 9.06.2018)

2 .NET Development [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/en-us/library/ff361664(v=vs.110).aspx. – Заглавие с экрана. – (Дата обращения: 12.06.2018)

3 Шарп, Дж. Microsoft Visual C#. Подробное руководство / Дж. Шарп. - 8-е издание — СПб.: Питер, 2017. — 848 с.

4 Культин, Н. Microsoft Visual C# в задачах и примерах C# в задачах и примерах / Н. Культин - М.: Изд-во БХВ-Петербург, 2015. — 314 с.

5 Удаление невидимых линий и поверхностей [Электронный ресурс]. – Режим доступа: http://compgraph.tpu.ru/Del_hide_line.htm. – Заглавие с экрана. – (Дата обращения: 9.06.2018)

6 Боресков, А. Компьютерная графика. Учебник и практикум / А. Боресков, Е. Шикин – М.: Изд-во Юрайт, 2016 – 220 с.

7 Windows Forms [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/en-us/dotnet/framework/winforms/index. – Заглавие с экрана. – (Дата обращения: 20.03.2018)

8 Chand. M. C# Timer [Электронный ресурс]/ M. Chand. – Режим доступа: https://www.c-sharpcorner.com/UploadFile/mahesh/C-Sharp-timer/. – (Дата обращения: 28.05.2018)

9 Chand. M. PictureBox in C# [Электронный ресурс]/ M. Chand. – Режим доступа: ttps://www.c-sharpcorner.com/uploadfile/mahesh/picturebox-in-C-Sharp/. – (Дата обращения: 15.04.2018)

10 Sells C. Windows Forms Programming in C#/ C. Sells – Addison Wesley, Boston. – 2005 – 744

11 Лабор, В.В. C# Создание приложений для Windows/ В.В. Лабор – М.: - Изд-во Харвест, Минск,2003 -385 с.

12 The Basics of Drawing Graphics onto Windows Forms [Электронный ресурс]– Режим доступа: https://www.c-sharpcorner.com/uploadfile/TheButler/the-basics-of-drawing-graphics-onto-windows-forms/ – (Дата обращения: 30.05.2018)

13 Изменяем внешний вид System.Windows.Forms.Form [Электронный ресурс] – Режим доступа: http://coolcode.ru/izmenyaem-vneshniy-vid-system-windows-forms-form/ – (Дата обращения: 03.03.2018)

14 Creating multiple windows with OpenGL and GLUT [Электронный ресурс] – Режим доступа: http://www.codeincodeblock.com/2012/05/creating-multiple-windows-with-opengl.html/ – (Дата обращения: 08.04.2018)

15 SharpGL [Электронный ресурс] – Режим доступа: https://sites.google.com/site/raznyeurokipoinformatiki/home/c/sharpgl – (Дата обращения: 25.05.2018)
16 Удаление невидимых поверхностей [Электронный ресурс] – Режим доступа: http://www.codenet.ru/progr/alg/del.php– (Дата обращения: 10.06.2018)
17 Разновидности компьютерной графики [Электронный ресурс] – Режим доступа: http://helpiks.org/2-33831.html– (Дата обращения: 15.05.2018)
18 Сравнение операционных систем Mac OS, Linux и Windows [Электронный ресурс] – Режим доступа: http://suhorukov.com/news_akademy/sravnenie-operacionnyh-sistem-mac-os-linux-i-windows – (Дата обращения: 25.02.2018)
19 Сравнительный анализ DirectX 11, OpenGL и Vulkan. [Электронный ресурс]. - Режим доступа: http://www.hardwareluxx.ru/index.php/artikel/hardware/grafikkarten/37520-directx-11-vulkan-opengl.html. - (дата обращения 01.04.2018).
20 Обзор программ обучения направления “Программирование на C++ и компьютерная графика” [Электронный ресурс]. - Режим доступа: http://haidarovg.narod.ru/. - (дата обращения 10.02.2018).
21 Ярош Е.С. Методы и средства представления графической информации/ Е.С. Ярош. - 226 с.

```
myWind = new Pen(Color.White, 1);
```

Видимая часть первого прямоугольника.

```
g.DrawLine(myWind, new Point(200, 150), new Point(800, 150));
g.DrawLine(myWind, new Point(200, 150), new Point(300, 100));
g.DrawLine(myWind, new Point(300, 100), new Point(573, 100));
g.DrawLine(myWind, new Point(900, 100), new Point(808, 100));
g.DrawLine(myWind, new Point(800, 150), new Point(800, 450));
g.DrawLine(myWind, new Point(800, 150), new Point(900, 100));
g.DrawLine(myWind, new Point(800, 450), new Point(200, 450));
g.DrawLine(myWind, new Point(900, 100), new Point(900, 400));
g.DrawLine(myWind, new Point(800, 450), new Point(900, 400));
g.DrawLine(myWind, new Point(200, 150), new Point(200, 450));
```

Видимая часть второго прямоугольника.

```
g.DrawLine(myWind, new Point(800, 110), new Point(740, 140));
g.DrawLine(myWind, new Point(740, 140), new Point(540, 140));
g.DrawLine(myWind, new Point(840, 60), new Point(780, 90));
g.DrawLine(myWind, new Point(780, 90), new Point(580, 90));
g.DrawLine(myWind, new Point(840, 60), new Point(640, 60));
g.DrawLine(myWind, new Point(640, 60), new Point(580, 90));
g.DrawLine(myWind, new Point(800, 110), new Point(840, 60));
g.DrawLine(myWind, new Point(740, 140), new Point(780, 90));
g.DrawLine(myWind, new Point(540, 140), new Point(580, 90));
g.DrawLine(myWind, new Point(220, 540), new Point(420, 540));
g.DrawLine(myWind, new Point(292, 450), new Point(220, 540));
g.DrawLine(myWind, new Point(492, 450), new Point(420, 540));
g.DrawLine(myWind, new Point(480, 510), new Point(420, 540));
g.DrawLine(myWind, new Point(480, 510), new Point(528, 450));
g.DrawLine(myWind, new Point(325, 285), new Point(325, 10));
g.DrawLine(myWind, new Point(310, 275), new Point(1090, 275));
g.DrawLine(myWind, new Point(340, 265), new Point(-150, 550));
```

```
          for (temp_1 = 100; temp_1 < 400;)
          {
             g.DrawLine(myWind, new Point(300, temp_1), new Point(300, temp_1 +
10));
             temp_1 = temp_1 + 20;
          }
          for (temp_1 = 576; temp_1 < 810;)
          {
             g.DrawLine(myWind, new Point(temp_1, 100), new Point(temp_1 + 10,
100));
             temp_1 = temp_1 + 20;
          }
          for (temp_1 = 300; temp_1 < 900;)
          {
             g.DrawLine(myWind, new Point(temp_1, 400), new Point(temp_1 + 10,
400));
             temp_1 = temp_1 + 20;
          }
          temp_2 = 400;
          for (temp_1 = 300; temp_1 > 200;)
          {

             g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1 - 10,
temp_2 + 5));
             temp_1 = temp_1 - 20;
             temp_2 = temp_2 + 10;
          }
          for (temp_1 = 300; temp_1 < 500;)
          {
             g.DrawLine(myWind, new Point(temp_1, 440), new Point(temp_1 + 10,
440));
             temp_1 = temp_1 + 20;
          }
          for (temp_1 = 280; temp_1 < 480;)
          {
             g.DrawLine(myWind, new Point(temp_1, 510), new Point(temp_1 + 10,
510));
             temp_1 = temp_1 + 20;
          }
          for (temp_1 = 360; temp_1 < 560;)
          {
             g.DrawLine(myWind, new Point(temp_1, 410), new Point(temp_1 + 10,
410));
```

```
            temp_1 = temp_1 + 20;
        }
        for (temp_1 = 600; temp_1 < 800;)
        {
            g.DrawLine(myWind, new Point(temp_1, 110), new Point(temp_1 + 10,
110));
            temp_1 = temp_1 + 20;
        }
        temp_2 = 510;
        for (temp_1 = 280; temp_1 > 220;)
        {
            g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1 - 10,
temp_2 + 5));
            temp_1 = temp_1 - 20;
            temp_2 = temp_2 + 10;
        }
        temp_2 = 410;
        for (temp_1 = 560; temp_1 > 500;)
        {
            g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1 - 10,
temp_2 + 5));
            temp_1 = temp_1 - 20;
            temp_2 = temp_2 + 10;
        }
        temp_2 = 140;
        for (temp_1 = 740; temp_1 > 492;)
        {
            g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1 - 8,
temp_2 + 10));
            temp_1 = temp_1 - 16;
            temp_2 = temp_2 + 20;
        }
        temp_2 = 140;
        for (temp_1 = 540; temp_1 > 292;)
        {
            g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1 - 8,
temp_2 + 10));
            temp_1 = temp_1 - 16;
            temp_2 = temp_2 + 20;
        }
        temp_2 = 60;
        for (temp_1 = 640; temp_1 > 280;)
        {
```

```
            g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1 - 8,
temp_2 + 10));
            temp_1 = temp_1 - 16;
            temp_2 = temp_2 + 20;
        }
        temp_2 = 110;
        for (temp_1 = 800; temp_1 > 528;)
        {
            g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1 - 8,
temp_2 + 10));
            temp_1 = temp_1 - 16;
            temp_2 = temp_2 + 20;
        }
        temp_2 = 410;
        for (temp_1 = 360; temp_1 > 300;)
        {
            g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1 - 10,
temp_2 + 5));
            temp_1 = temp_1 - 20;
            temp_2 = temp_2 + 10;
        }
        temp_2 = 110;
        for (temp_1 = 600; temp_1 > 540;)
        {
            g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1 - 10,
temp_2 + 5));
            temp_1 = temp_1 - 20;
            temp_2 = temp_2 + 10;
        }
```

```
            if (temp < 5)
            {

            }
            else if (temp < 10)
            {
               for (temp_1 = 100; temp_1 < 400;)
               {
                  g.DrawLine(myWind, new Point(300, temp_1), new Point(300,
temp_1 + 10));
                  temp_1 = temp_1 + 20;
               }
            }
            else if (temp < 15)
            {
               for (temp_1 = 300; temp_1 < 900;)
               {
                  g.DrawLine(myWind, new Point(temp_1, 400), new Point(temp_1 +
10, 400));
                  temp_1 = temp_1 + 20;
               }
            }
            else if (temp < 20)
            {
               temp_2 = 400;
               for (temp_1 = 300; temp_1 > 200;)
               {
                  g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1
- 10, temp_2 + 5));
                  temp_1 = temp_1 - 20;
                  temp_2 = temp_2 + 10;
               }
            }
            else if (temp < 25)
            {
               temp_2 = 60;
               for (temp_1 = 640; temp_1 > 280;)
               {
                  g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1
- 8, temp_2 + 10));
                  temp_1 = temp_1 - 16;
                  temp_2 = temp_2 + 20;
               }
```

```
                myWind = new Pen(Color.White, 1);
                temp_2 = 120;
                for (temp_1 = 592; temp_1 > 280;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1
- 8, temp_2 + 10));
                    temp_1 = temp_1 - 16;
                    temp_2 = temp_2 + 20;
                }
            }
            else if (temp < 30)
            {
                for (temp_1 = 600; temp_1 < 800;)
                {
                    g.DrawLine(myWind, new Point(temp_1, 110), new Point(temp_1 +
10, 110));
                    temp_1 = temp_1 + 20;
                }

            }
            else if (temp < 35)
            {
                temp_2 = 110;
                for (temp_1 = 600; temp_1 > 540;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1
- 10, temp_2 + 5));
                    temp_1 = temp_1 - 20;
                    temp_2 = temp_2 + 10;
                }
            }
            else if (temp < 40)
            {
                temp_2 = 140;
                for (temp_1 = 540; temp_1 > 292;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1
- 8, temp_2 + 10));
                    temp_1 = temp_1 - 16;
                    temp_2 = temp_2 + 20;
                }
                myWind = new Pen(Color.White, 1);
                g.DrawLine(myWind, new Point(300, 440), new Point(292, 450));
            }
```

```
            else if (temp < 45)
            {
                temp_2 = 120;
                for (temp_1 = 592; temp_1 > 280;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1
- 8, temp_2 + 10));
                    temp_1 = temp_1 - 16;
                    temp_2 = temp_2 + 20;
                }
                myWind = new Pen(Color.White, 1);
                temp_2 = 420;
                for (temp_1 = 352; temp_1 > 280;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1
- 8, temp_2 + 10));
                    temp_1 = temp_1 - 16;
                    temp_2 = temp_2 + 20;
                }
            }
            else if (temp < 50)
            {
                temp_2 = 140;
                for (temp_1 = 740; temp_1 > 492;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1
- 8, temp_2 + 10));
                    temp_1 = temp_1 - 16;
                    temp_2 = temp_2 + 20;
                }
                myWind = new Pen(Color.White, 1);
                g.DrawLine(myWind, new Point(500, 440), new Point(492, 450));
            }
            else if (temp < 55)
            {
                temp_2 = 110;
                for (temp_1 = 800; temp_1 > 528;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1
- 8, temp_2 + 10));
                    temp_1 = temp_1 - 16;
                    temp_2 = temp_2 + 20;
                }
                myWind = new Pen(Color.White, 1);
```

```
                temp_2 = 410;
                for (temp_1 = 560; temp_1 > 528;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1
- 8, temp_2 + 10));
                    temp_1 = temp_1 - 16;
                    temp_2 = temp_2 + 20;
                }
            }
            else if (temp < 65)
            {
                temp_2 = 410;
                for (temp_1 = 360; temp_1 > 300;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1
- 10, temp_2 + 5));
                    temp_1 = temp_1 - 20;
                    temp_2 = temp_2 + 10;
                }

            }
            else if (temp < 70)
            {
                for (temp_1 = 300; temp_1 < 560;)
                {
                    g.DrawLine(myWind, new Point(temp_1, 410), new Point(temp_1 +
10, 410));
                    temp_1 = temp_1 + 20;
                }

            }
            else if (temp < 75)
            {
                temp_2 = 60;
                for (temp_1 = 640; temp_1 > 280;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1
- 8, temp_2 + 10));
                    temp_1 = temp_1 - 16;
                    temp_2 = temp_2 + 20;
                }
            }
            else if (temp < 80)
            {
```

```
                temp_2 = 510;
                for (temp_1 = 280; temp_1 > 220;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new Point(temp_1
- 10, temp_2 + 5));
                    temp_1 = temp_1 - 20;
                    temp_2 = temp_2 + 10;
                }
            }
            else if (temp < 85)
            {
                for (temp_1 = 280; temp_1 < 480;)
                {
                    g.DrawLine(myWind, new Point(temp_1, 510), new Point(temp_1 +
10, 510));
                    temp_1 = temp_1 + 20;
                }
            }
            temp++;
```

```
                if (temp < 510)
                {
                   for (x = 0; x < 1150; x++)
                   {

                      col = (pictureBox7.Image as Bitmap).GetPixel(x, temp -80);
                      if (col.ToArgb() == Color.White.ToArgb())
                      {
                         (pictureBox7.Image as Bitmap).SetPixel(x, temp -80,
Color.Black);
                      }
                   }

                   temp++;
                }
                else if (temp == 600)
                {
                   temp = 0;
                }
```

```
if (temp < 320)
                {
                   x = temp + 486;
                   myWind = new Pen(Color.Black, 1);

                   for (temp_1 = x; temp_1 < 810;)
                   {
                      g.DrawLine(myWind, new Point(temp_1, 100), new Point(temp_1
+ 10, 100));
                      temp_1 = temp_1 + 20;
                   }
                   myWind = new Pen(Color.White, 1);

                   for (temp_1 = x+1; temp_1 < 810;)
                   {
                      g.DrawLine(myWind, new Point(temp_1, 100), new Point(temp_1
+ 10, 100));
                      temp_1 = temp_1 + 20;
                   }
                   temp++;
                }
                else if (temp == 320)
                {
                   x = temp + 490;
                   myWind = new Pen(Color.Black, 1);

                   for (temp_1 = x; temp_1 < 810;)
                   {
                      g.DrawLine(myWind, new Point(temp_1, 100), new Point(temp_1
+ 10, 100));
                      temp_1 = temp_1 + 20;
                   }
                   myWind = new Pen(Color.White, 1);

                   for (temp_1 = x + 1; temp_1 < 810;)
                   {
                      g.DrawLine(myWind, new Point(temp_1, 100), new Point(temp_1
+ 1, 100));
                      temp_1 = temp_1 + 2;
                   }
                   temp++;
                }
                else if (temp <330)
```

```
                {
                    x = temp + 490;
                    myWind = new Pen(Color.Black, 1);

                    for (temp_1 = x; temp_1 < 810;)
                    {
                        g.DrawLine(myWind, new Point(temp_1, 100), new Point(temp_1
+ 1, 100));
                        temp_1 = temp_1 + 2;
                    }
                    myWind = new Pen(Color.White, 1);

                    for (temp_1 = x + 1; temp_1 < 810;)
                    {
                        g.DrawLine(myWind, new Point(temp_1, 100), new Point(temp_1
+ 1, 100));
                        temp_1 = temp_1 + 2;
                    }
                    temp++;
                }
                else if (temp < 340)
                {
                    for (x = 0; x < 1150; x++)
                    {
                        col = (pictureBox7.Image as Bitmap).GetPixel(x, temp + 110);
                        if (col.ToArgb() == Color.White.ToArgb())
                        {
                            (pictureBox7.Image as Bitmap).SetPixel(x, temp + 110,
Color.Black);
                        }
                    }
                    myWind = new Pen(Color.White, 1);
                    temp_2 = 410;
                    for (temp_1 = 560; temp_1 > 500;)
                    {
                        g.DrawLine(myWind, new Point(temp_1, temp_2), new
Point(temp_1 - 10, temp_2 + 5));
                        temp_1 = temp_1 - 20;
                        temp_2 = temp_2 + 10;
                    }
                    for (temp_1 = 300; temp_1 < 500;)
                    {
                        g.DrawLine(myWind, new Point(temp_1, 440), new Point(temp_1
+ 10, 440));
```

```
                    temp_1 = temp_1 + 20;
                }
                temp_2 = 410;
                for (temp_1 = 560; temp_1 > 528;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new
Point(temp_1 - 8, temp_2 + 10));
                    temp_1 = temp_1 - 16;
                    temp_2 = temp_2 + 20;
                }
                temp_2 = 440;
                for (temp_1 = 500; temp_1 > 492;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new
Point(temp_1 - 8, temp_2 + 10));
                    temp_1 = temp_1 - 16;
                    temp_2 = temp_2 + 20;
                }
                temp++;
            }
            else if (temp < 540)
            {
                x = temp -40;
                myWind = new Pen(Color.Black, 1);

                for (temp_1 = x; temp_1 < 500;)
                {
                    g.DrawLine(myWind, new Point(temp_1, 440), new Point(temp_1
+ 10, 440));
                    temp_1 = temp_1 + 20;
                }
                myWind = new Pen(Color.White, 1);

                for (temp_1 = x + 1; temp_1 < 500;)
                {
                    g.DrawLine(myWind, new Point(temp_1, 440), new Point(temp_1
+ 10, 440));
                    temp_1 = temp_1 + 20;
                }
                temp++;
            }
            else if (temp < 550)
            {
                for (x = 0; x < 1150; x++)
```

```
                {
                    col = (pictureBox7.Image as Bitmap).GetPixel(x, temp - 100);
                    if (col.ToArgb() == Color.White.ToArgb())
                    {
                        (pictureBox7.Image as Bitmap).SetPixel(x, temp - 100,
Color.Black);
                    }
                }
                myWind = new Pen(Color.White, 1);
                temp_2 = 410;
                for (temp_1 = 560; temp_1 > 500;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new
Point(temp_1 - 10, temp_2 + 5));
                    temp_1 = temp_1 - 20;
                    temp_2 = temp_2 + 10;
                }
                temp_2 = 410;
                for (temp_1 = 560; temp_1 > 528;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new
Point(temp_1 - 8, temp_2 + 10));
                    temp_1 = temp_1 - 16;
                    temp_2 = temp_2 + 20;
                }
                temp++;
            }
            else if (temp < 610)
            {
                for (x = 0; x < 1150; x++)
                {
                    col = (pictureBox7.Image as Bitmap).GetPixel(x, 990 - temp);
                    if (col.ToArgb() == Color.White.ToArgb())
                    {
                        (pictureBox7.Image as Bitmap).SetPixel(x, 990 - temp,
Color.Black);
                    }
                }
                temp_2 = 410;
                for (temp_1 = 560; temp_1 > 528;)
                {
                    g.DrawLine(myWind, new Point(temp_1, temp_2), new
Point(temp_1 - 8, temp_2 + 10));
                    temp_1 = temp_1 - 16;
```

```
                temp_2 = temp_2 + 20;
            }
            temp++;
        }
        else if (temp < 645)
        {
            for (x = 0; x < 1150; x++)
            {
                col = (pictureBox7.Image as Bitmap).GetPixel(x, temp - 200);
                if (col.ToArgb() == Color.White.ToArgb())
                {
                    (pictureBox7.Image as Bitmap).SetPixel(x, temp - 200,
Color.Black);
                }
            }

            temp++;
        }
        else if (temp==645)
        {
            temp = 0;
                        }
```