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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка программы построения 3Д сцен помещений различной планировки»

Студент ИУ7-55Б _____ О.Н. Талышева
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ Н.Н. Мартынюк
(Подпись, дата) (И.О.Фамилия)

2024 г

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

УТВЕРЖДАЮ

Заведующий кафедрой ИУ7
(индекс)
И.В.Рудаков
(И.О. Фамилия)
(подпись)
(дата)

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине Компьютерная графика
Студент группы ИУ7-45Б Талышева Олеся Николаевна
(Фамилия, имя, отчество)

Тема курсовой работы
Разработка программы построения 3Д сцен помещений различной планировки.

Направленность КР (учебная, исследовательская, практическая, производственная, др.)
практическая

Источник тематики (кафедра, предприятие, НИР) предприятие

Задание Разработать программу построения 3Д сцен помещений различной планировки. Создать объекты (стена, окно, дверь), которые можно добавлять на сцену, удалять со сцены, перемещать, поворачивать и масштабировать на сцене. Предоставить возможность рассмотреть сцену из разных точек с помощью камеры. Обеспечить сохранение модели в файл и загрузку существующей для последующего редактирования.

Оформление курсовой работы:
2.1. Расчетно-пояснительная записка на 25–30 листах формата А4.
Расчетно-пояснительная записка должна содержать постановку введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.
2.2. Перечень графического материала (плакаты, схемы, чертежи и т. п.) На защиту проекта должна быть представлена презентация, состоящая из 15–20 слайдов.

Дата выдачи задания « ___ » _____ 20__ г.

Руководитель курсовой работы _____ Мартынюк Н.Н.
(подпись, дата) (И.О. Фамилия)

Студент _____ Талышева О.Н.
(подпись, дата) (И.О. Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Содержание

ВВЕДЕНИЕ

В настоящее время введение технологий 3D моделирования активно развивается в самых разных сферах: архитектуре, дизайне, инженерии, безопасности и оперативном реагировании. Программное обеспечение для построения 3D сцен позволяет визуализировать объекты и пространства, что существенно облегчает их анализ, планирование и эксплуатацию. В особенности, технологии моделирования помещений с гибкой настройкой планировок и объектов находят важное применение для задач охраны и контроля доступа на стратегически важные объекты.

Одной из ключевых сфер, в которой данная тема является особенно актуальной, выступает обеспечение безопасности и оперативного реагирования силовых структур на проникновение на охраняемые объекты. В условиях возросших угроз безопасности и повышенных требований к защите территорий и объектов, необходимость быстрого создания точных моделей помещений с целью их анализа является приоритетной задачей. Использование 3D моделей позволяет оперативно прорабатывать сценарии вторжений, рассчитывать оптимальные пути пресечения проникновений и создавать план эвакуации или нейтрализации угрозы.

Для силовых структур важно иметь инструменты, которые позволяют в режиме реального времени моделировать различные планировки и адаптировать стратегические планы по обеспечению безопасности. Программа, позволяющая моделировать помещения с возможностью добавления объектов (стен, окон, дверей), их перемещения, масштабирования и управления камерой, может стать незаменимым инструментом для подготовки к операциям, тренировок и планирования реагирования в критических ситуациях.

Таким образом, разработка программы построения 3D сцен помещений имеет важное практическое значение, поскольку позволяет оперативно создавать модели для анализа и разработки решений по обеспечению безопасности.

Целью данной курсовой работы является разработка программного обеспечения для создания и редактирования 3D сцен помещений с возможностью интерактивного добавления объектов (стен, окон, дверей), их перемещения, масштабирования, поворота, а также обеспечения сохранения и загрузки моделей.

В рамках работы были поставлены следующие задачи:

1. Анализ требований к программе и исследование существующих решений:
   (a) Изучить программные продукты для 3D моделирования помещений, чтобы понять их функциональные особенности и интерфейсные решения.
   (b) Оценить, какие элементы и функции наиболее важны для конечного пользователя.
2. Изучить алгоритмы реализации технических решений и выбрать наиболее подходящие для работы с 3D сценами.

3. Разработка архитектуры программы:

   (a) Спроектировать структуру программы, определив основные компоненты: объекты сцены (стены, окна, двери), камера, управление сценой.

   (b) Разработать систему хранения данных о 3D моделях, обеспечивающую сохранение и загрузку сцены.

4. Реализация объектов для создания сцены:

   (a) Создать базовые 3D объекты (стена, окно, дверь) с параметрами (размеры, позиции, углы поворота, текстуры).

   (b) Обеспечить возможность добавления, перемещения, удаления, масштабирования и поворота объектов на сцене.

5. Разработка системы управления камерой:

   (a) Предоставить пользователю возможность управления камерой для осмотра сцены под разными углами.

   (b) Реализовать функции перемещения камеры, вращения вокруг объектов, изменения масштаба.

6. Реализация пользовательского интерфейса (UI):

   (a) Разработать удобный интерфейс для добавления и редактирования объектов сцены.

   (b) Включить панели инструментов для выбора объектов, изменения их параметров, управления сценой и камерой.

7. Сохранение и загрузка 3D сцен:

   (a) Реализовать функционал сохранения текущей сцены в файл в специальном формате, чтобы пользователи могли продолжить работу позже.

   (b) Предусмотреть возможность загрузки ранее сохранённых сцен для редактирования.

8. Тестирование программы:

   (a) Провести тестирование работы программы для различных вариантов планировки помещений.

   (b) Проверить корректность работы с сохранением и загрузкой сцен, взаимодействие с объектами и камерой.

9. Оценка производительности программы:

(a) Провести анализ производительности программы при увеличении количества объектов на сцене.

(b) Оптимизировать работу с 3D объектами для плавного взаимодействия даже при больших сценах.

10. Документирование и подготовка отчётной документации:

(a) Описать процесс разработки, результаты тестирования, а также подготовить руководство пользователя для программы.

1. Аналитическая часть

В данном разделе будут представлены описание объектов сцены, а также обоснование выбора алгоритмов, которые будут использоваться для визуализации.

1.1. Описание объектов сцены

Сцена состоит из различных объектов, включая стены, окна и двери. Каждый объект может быть представлен в виде параллелепипеда с заданными параметрами длины, ширины и высоты. Особое место среди объектов занимает сама сцена – это плоскость, состоящая из заданного пользователем количества квадратов, размерами которых измеряются следующие объекты и относительно которых задаётся расположение стен, окон и дверей на экране.

1. Стена – это параллелепипед с заданными размерами, ограничивающий пространство. Стены могут быть различной толщины и высоты и по разному располагаться на сцене в зависимости от характеристик помещения. Эти объекты используются для формирования границ помещения.

2. Окно – это плоский объект, встроенный в стену. Его размеры могут варьироваться в зависимости от проекта. Окна также могут быть расположены на разных уровнях стен.

3. Дверь – это параллелепипед, также встроенный в стену, с возможностью регулирования размера и положения.

4. Источник света – это точка, расположенная где-то далеко в бесконечности. Он излучает световые лучи во все стороны, что позволяет освещать сцену и создавать тени. Положение источника света можно изменять для достижения различных визуальных эффектов и освещенности объектов сцены.

5. Камера – представляет собой объект, который может вращаться и рассматривать сцену с разных сторон и расстояний. Камера позволяет настраивать угол обзора, масштаб и положение относительно объектов сцены, что даёт возможность пользователю детально осмотреть помещение.

Эти объекты могут иметь различные параметры, что позволяет создавать уникальные конфигурации помещений, соответствующие потребностям пользователя и требованиям дизайна.

1.2. Выбор формы задания трёхмерных моделей

В процессе 3D моделирования создаются геометрические модели, т.е. модели, отражающие геометрические свойства изделий. Различают геометрические модели каркасные (проволочные), поверхностные, объемные (твердотельные).

1. Каркасная модель представляет форму детали в виде конечного множества линий, лежащих на поверхностях детали. Для каждой линии известны координаты концевых точек и указана их инцидентность рёбрам или поверхностям.

2. Поверхностная модель отображает форму детали с помощью задания ограничивающих её поверхностей, например, в виде совокупности данных о гранях, рёбрах и вершинах. Возможны различные виды задания поверхностей — плоскости, поверхности вращения, линейчатые поверхности.

3. Объёмная модель позволяет представить сложные изделия с обеспечением логической связанности информации, в частности, благодаря введению понятия о материале. В такой модели хранится информация, позволяющая отличать материал от пустоты, при этом пустота может рассматриваться как особый вид материала. [8]

В процессе 3D-моделирования выбор геометрической модели зависит от целей и характеристик задачи. Каркасные модели предоставляют лишь ограниченную информацию о форме объекта, что не всегда подходит для детализированного представления. Объёмные модели дают полное представление о внутреннем содержании, но они требуют значительно больше вычислительных ресурсов для обработки.

В рамках данной работы была выбрана поверхностная модель. Этот выбор обусловлен тем, что поверхностные модели позволяют эффективно представлять форму объекта с помощью ограничивающих его поверхностей, что идеально подходит для визуализации объектов на сцене. Поверхностная модель даёт возможность адекватно отобразить внешний вид объектов (стен, окон, дверей), при этом её использование не требует таких значительных вычислительных затрат, как объёмные модели. Такая модель также достаточно гибка для выполнения операций, связанных с рендерингом и проверкой взаимодействий объектов на сцене.

1.3. Выбор алгоритма удаления невидимых линий

1.3.1 Введение

Одной из основных задач компьютерной графики является визуализация трёхмерных сцен. Подобные задачи возникают в системах автоматизированного проектирования, пакетах моделирования физических процессов, средствах компьютерной анимации и виртуальной реальности.

При отображении трёхмерной сцены на экране некоторые из объектов сцены могут заслонить другие объекты. Заслонённые части объектов невидимы и не должны рисоваться, или должны рисоваться иначе, чем видимые части, например, пунктиром. Если этого не делать, то изображение будет выглядеть неправильно.

Такие задачи решают с помощью алгоритмов удаления невидимых линий и поверхностей. Если сцена отображается в каркасном виде, линиями, то нужно удалять

невидимые линии. Каркасное изображение обычно строится из отрезков – рёбер, и алгоритм должен выделить части отрезков, заслонённых объектами сцены. Если объекты сцены отображаются в виде закрашенных поверхностей, то нужно удалять невидимые части этих поверхностей. Обычно в качестве поверхностей используются выпуклые многоугольники, чаще всего – треугольники. В курсовой работе необходимо реализовать один из алгоритмов удаления невидимых линий и поверхностей и произвести анализ его производительности. В соответствующих разделах приведены описания основных алгоритмов: Робертса, Варнака, Z-буфера, художника, трассировки лучей, построчного сканирования.

Все алгоритмы удаления невидимых линий и поверхностей можно разделить на две группы:

1. В одних, сначала определяется видимость для участков линий или поверхностей, а затем рисуются видимые части. К таким алгоритмам относится, например, алгоритм Робертса. В подобных алгоритмах необходимо сравнить каждый объект сцены (линию или поверхность) с остальными объектами, способными заслонить его.

2. В других, для каждого пиксела изображения определяется, какой из объектов сцены в нём виден. К таким алгоритмам относятся, например, алгоритмы трассировки лучей или Z-буфера. В этом случае нужно для каждого пиксела выбрать ближайший к наблюдателю объект сцены.

В обоих случаях требуется много вычислений, из-за того, что необходимо перебирать все объекты сцены. Для сокращения объёма вычислений используется свойство когерентности (англ. coherence – связность) расположенных рядом объектов. Можно выделить три вида когерентности:

1. Когерентность в картинной плоскости (плоскости экрана) – расположенные рядом пикселы, скорее всего, имеют одинаковые свойства, например, принадлежат одному и тому же объекту сцены, или видимы или, наоборот, заслонены одним и тем же объектом сцены.

2. Когерентность в объектном пространстве (пространстве сцены) – расположенные рядом объекты сцены, скорее всего, имеют одинаковые свойства, например, видимы или, наоборот, заслонены одним и тем же объектом сцены.

3. Когерентность во времени – при перемещении наблюдателя, на соседних кадрах будут видимы примерно одни и те же объекты сцены

В некоторых алгоритмах когерентность используется явно. Например, в алгоритме Варнака картинная плоскость рекурсивно делится на области расположенных рядом

пикселей, и, если вся область видима или, наоборот, заслонена, то дальнейшее её разделение не требуется. Другие алгоритмы можно модифицировать, используя свойство когерентности, существенно повышая их производительность. [3]

Существуют два различных способа изображения трехмерных тел — каркасное (wireframe — рисуются только ребра) и сплошное (рисуются закрашенные грани). Тем самым возникают два типа задач — удаление невидимых линий (ребер для каркасных изображений) и удаление невидимых поверхностей (граней для сплошных изображений).

Анализ видимости объектов можно производить как в исходном трехмерном пространстве, так и на картинной плоскости. Это приводит к разделению методов на два класса:

1. методы, работающие непосредственно в пространстве самих объектов;
2. методы, работающие в пространстве картинной плоскости, т. е. работающие с проекциями объектов.

Получаемый результат представляет собой либо набор видимых областей или отрезков, заданных с машинной точностью (имеет непрерывный вид), либо информацию о ближайшем объекте для каждого пиксела экрана (имеет дискретный вид).

Методы первого класса дают точное решение задачи удаления невидимых линий и поверхностей, никак не привязанное к растровым свойствам картинной плоскости.

Они могут работать как с самими объектами, выделяя те их части, которые видны, так и с их проекциями на картинную плоскость, выделяя на ней области, соответствующие проекциям видимых частей объектов, и, как правило, практически не привязаны к растровой решетке и свободны от погрешностей дискретизации. Так как эти методы работают с непрерывными исходными данными и получающиеся результаты не зависят от растровых свойств, то их иногда называют непрерывными методами (continuous methods).

Простейший вариант непрерывного подхода заключается в сравнении каждого объекта со всеми остальными, что дает временные затраты, пропорциональные n^2, где n — количество объектов в сцене.

Однако следует иметь в виду, что непрерывные методы, как правило, достаточно сложны.

Методы второго класса (point-sampling methods) дают приближенное решение задачи видимости, определяя видимость только в некотором наборе точек картинной плоскости — в точках растровой решетки. Они очень сильно привязаны к растровым свойствам картинной плоскости и фактически заключаются в определении для каждого пиксела той грани, которая является ближайшей к нему вдоль направления проектирования. Изменение разрешения приводит к необходимости полного перерасчета всего изображения.

Простейший вариант дискретного метода имеет временные затраты порядка Cn, где C — общее количество пикселов экрана, а n — количество объектов.

Всем методам второго класса традиционно свойственны ошибки дискретизации (aliasing artifacts). Однако, как правило, дискретные методы отличаются известной простотой.

Кроме этого существует довольно большое количество смешанных методов, использующих работу как в объектном пространстве, так и в картинной плоскости, методы, выполняющие часть работы с непрерывными данными, а часть — с дискретными.

Большинство алгоритмов удаления невидимых граней и поверхностей тесно связано с различными методами сортировки. Некоторые алгоритмы проводят сортировку явно, в некоторых она присутствует в скрытом виде. Приближенные методы отличаются друг от друга фактически только порядком и способом проведения сортировки.

Очень распространенной структурой данных в задачах удаления невидимых линий и поверхностей являются различные типы деревьев — двоичные (BSP-trees), четверичные (Quadtree), восьмеричные (Octree) и др.

Методы, практически применяющиеся в настоящее время, в большинстве являются комбинациями ряда простейших алгоритмов, неся в себе целый ряд разного рода оптимизаций. [4]

Основные методы оптимизации:

1. Метод отсечения нелицевых граней (culling)

   Позволяет примерно вдвое сократить количество рассматриваемых граней.

   Для определения того, является заданная грань лицевой или нет достаточно взять произвольную точку этой грани и проверить выполнение условия $(\mathbf{N},\mathbf{L}) \leq 0$, где $\mathbf{N}$ — нормаль к грани, $\mathbf{L}$ — направление проецирования.

2. Метод оболочек (Bounding Volumes, Bounding-Volume-Hierarchy (BVH))

   Если оболочки не пересекаются, то и содержащиеся в них объекты тоже пересекаться не будут. Однако, если оболочки пересекаются, то сами объекты пересекаться не обязаны. В качестве ограничивающих тел чаще всего используются прямоугольные ограничивающие параллелепипеды. Оболочка описывается числами $(\mathrm{Xmin},\mathrm{Ymin},\mathrm{Zmin})$ и $(\mathrm{Xmax},\mathrm{Ymax},\mathrm{Zmax})$ из координат точек исходного объекта (4 для 2D).

3. Разбиение пространства (картинной плоскости)

   Еще один метод, облегчающий сравнение объектов, позволяющий использовать когерентность как в пространстве, так и в картинной плоскости.

   С этой целью разбиения стоятся уже на этапе пре-процессирования, и для каждой клетки разбиения составляется список всех объектов (граней), которые ее пересекают.

Простейшим вариантом разбиения является равномерное разбиение пространства на набор равных прямоугольных клеток (см. рисунок 1). Составляется список объектов, пересекающих клетку разбиения. Для отыскания всех объектов, которые закрывают рассматриваемый объект при проецировании, проверяются только объекты, попадающие в те же клетки разбиения картинной плоскости.

Для сцен с неравномерным распределением объектов имеет смысл использовать неравномерное (адаптивное) разбиение пространства или плоскости (см. рисунок 1).

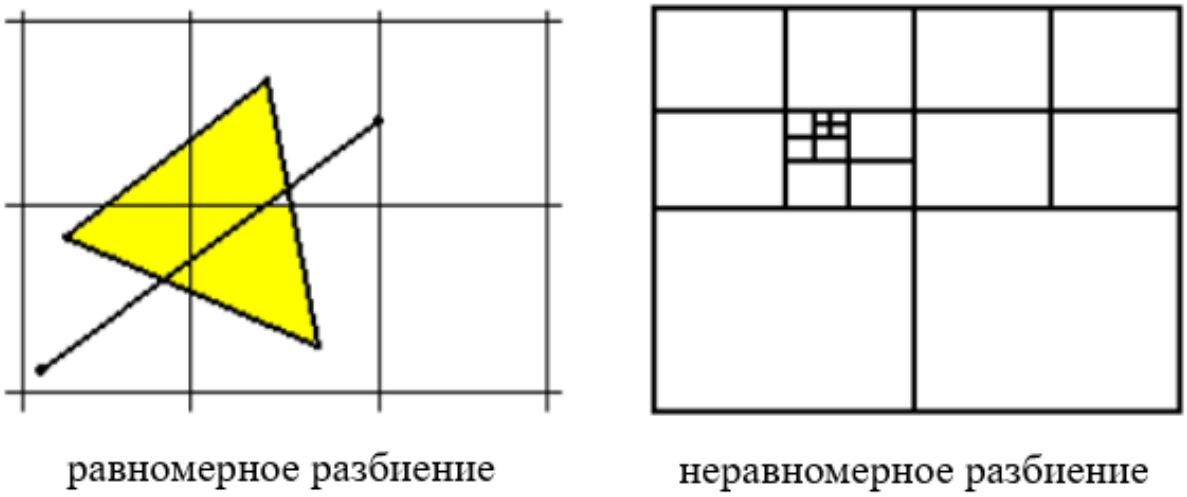

Рисунок 1 – Равномерное и неравномерное разбиения пространства объектов

4. Иерархические древовидные структуры

   При работе с большими объемами используются различные древовидные (иерархические) структуры. Стандартными формами таких структур являются восьмеричные (Octrees, для 3D), тетрарные (Quadtrees, для 2D), бинарные или BSP-деревья (Binary Space Partitioning Trees) и деревья ограничивающих тел. Иерархии позволяют упорядочивать грани объектов, производить быстрое и эффективное отсечение граней, не удовлетворяющих каким-либо из условий.

   (a) Иерархия ограничивающих тел

   Получается дерево, корнем которого является тело, описанное вокруг всей сцены, а потомками – тела, описанные вокруг первичных, вторичных и др. групп.

   Отсечение основного количества объектов происходит уже на ранней стадии достаточно быстро, ценой всего лишь нескольких проверок.

   (b) Иерархии разбиения

   Каждая клетка исходного разбиения разбивается на части (которые, в свою очередь, так же могут быть разбиты и т.д. При этом каждая клетка разбиения соответствует узлу дерева).

Иерархии (как и разбиения) позволяют достаточно легко и просто производить частичное упорядочение граней. В результате получается список граней, практически полностью упорядоченный, что дает возможность применять специальные методы сортировки.

Специальные методы оптимизации:

1. Потенциально видимые множества граней (препроцессинг построения PVS)
2. Метод порталов (PVS "на ходу")
3. Метод иерархических подсцен (модификация метода порталов) [5]

Ниже будут рассмотрены основные алгоритмы удаления евидимых линий.

1.3.2 Алгоритм плавающего горизонта

Алгоритм плавающего горизонта чаще всего используется для удаления невидимых линий трёхмерного представления функций, описывающих поверхность в виде:

$$F(x,y,z) = 0.$$

Подобные функции возникают во многих приложениях в математике, технике, естественных науках и других дисциплинах.

Предложено много алгоритмов, использующих этот подход. Поскольку в приложениях в основном интересуются описанием поверхности, этот алгоритм обычно работает в пространстве изображения. Главная идея данного метода заключается в сведении трехмерной задачи к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат x, y или z.

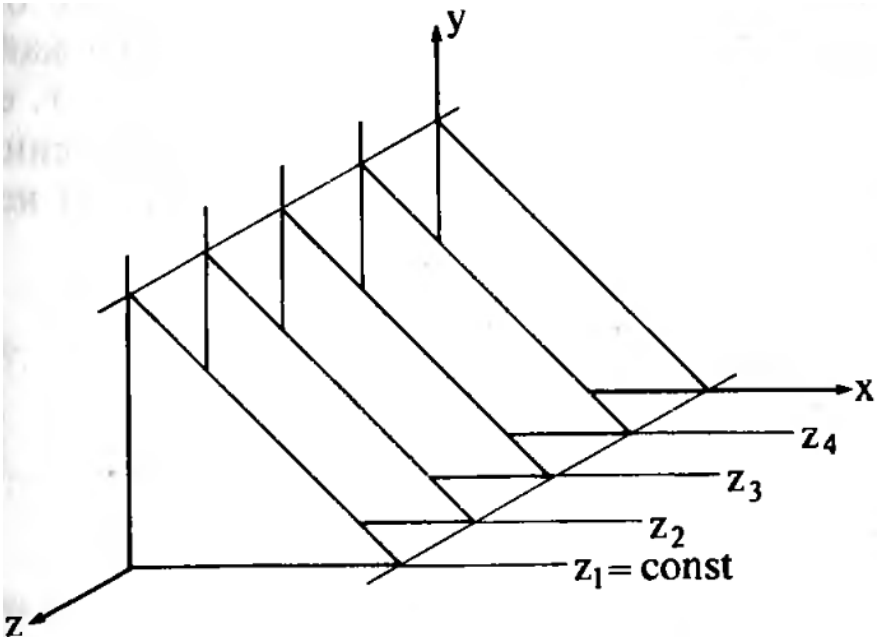

Рисунок 2 – Секущие плоскости с постоянной координатой.

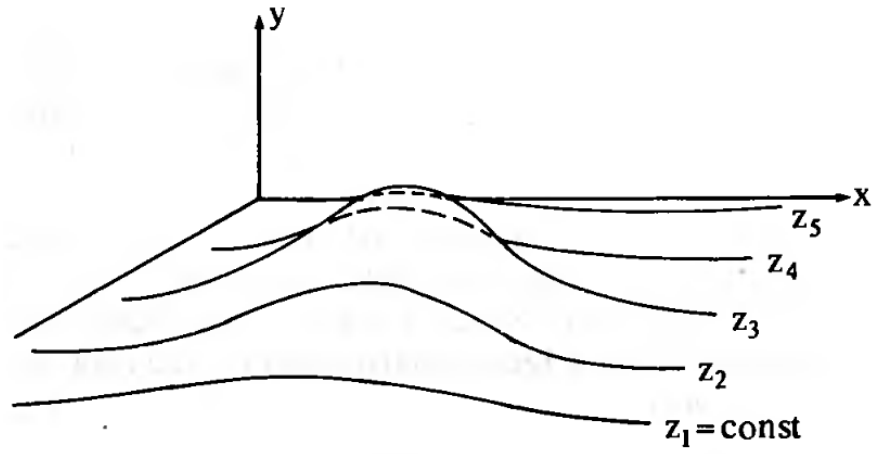

Рисунок 3 – Кривые в секущих плоскостях с постоянной координатой.

На рис. 2 приведен пример, где указанные параллельные плоскости определяются постоянными значениями z. Функция $F(x,y,z)=0$ сводится к последовательности кривых, лежащих в каждой из этих параллельных плоскостей, например, к последовательности

$$y = f(x,z) \quad \text{или} \quad x = g(y,z)$$

где z постоянно на каждой из заданных параллельных плоскостей.

Итак, поверхность теперь складывается из последовательности кривых, лежащих в каждой из этих плоскостей, как показано на рис. 3. Здесь предполагается, что полученные кривые являются однозначными функциями независимых переменных. Если спроецировать полученные кривые на плоскость $z=0$, как показано на рис. 4, то сразу становится ясна идея алгоритма удаления невидимых участков исходной поверхности.

Алгоритм сначала упорядочивает плоскости $z = \text{const}$ по возрастанию расстояния до них от точки наблюдения. Затем для каждой плоскости, начиная с ближайшей к точке наблюдения, строится кривая, лежащая на ней, т.е. для каждого значения координаты x в пространстве изображения определяется соответствующее значение y. Алгоритм удаления невидимой линии заключается в следующем:

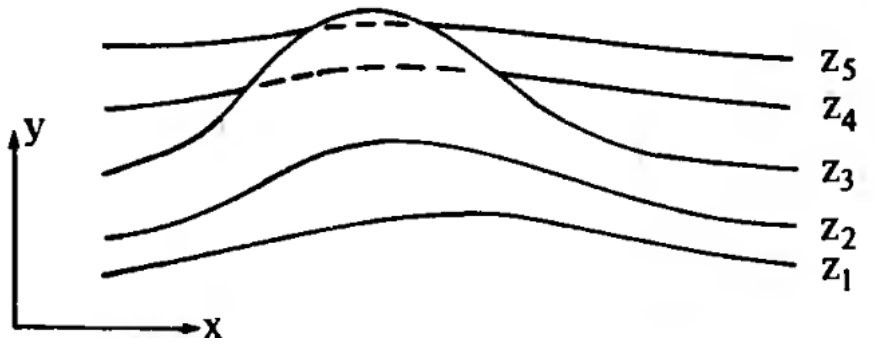

Рисунок 4 – Проекция кривых на плоскость z = 0.

Если на текущей плоскости при некотором заданном значении x соответствующее значение y на кривой больше значения y для всех предыдущих кривых при этом значении x, то текущая кривая видима в этой точке; в противном случае она невидима.

Невидимые участки показаны пунктиром на рис. 4. Реализация данного алгоритма достаточно проста. Для хранения максимальных значений y при каждом значении x используется массив, длина которого равна числу различимых точек (разрешению) по оси x в пространстве изображения. Значения, хранящиеся в этом массиве, представляют собой текущие значения «горизонта». Поэтому по мере рисования каждой очередной кривой этот горизонт «всплывает». Фактически этот алгоритм удаления невидимых линий работает каждый раз с одной линией. Алгоритм работает очень хорошо до тех пор, пока какая-нибудь очередная кривая не окажется ниже самой первой из кривых, как показано на рис. 5 а.

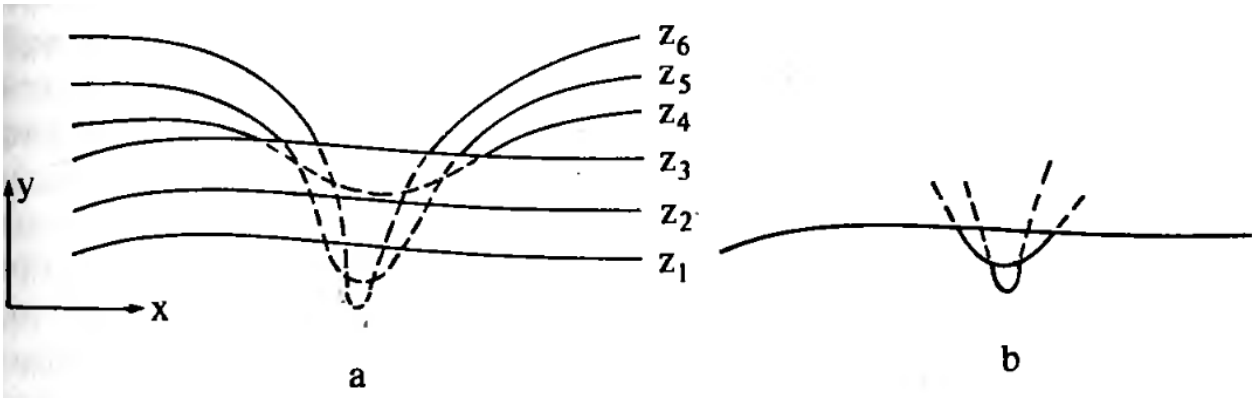

Рисунок 5 – Обработка нижней стороны поверхности.

Подобные кривые, естественно, видимы и представляют собой нижнюю сторону исходной поверхности, однако алгоритм будет считать их невидимыми. Нижняя сторона

поверхности делается видимой, если модифицировать этот алгоритм, включив в него нижний горизонт, который опускается вниз по ходу работы алгоритма. Это реализуется при помощи второго массива, длина которого равна числу различимых точек по оси x в пространстве изображения. Этот массив содержит наименьшие значения y для каждого значения x. Алгоритм теперь становится таким:

Если на текущей плоскости при некотором заданном значении x соответствующее значение y на кривой больше максимума или меньше минимума по y для всех предыдущих кривых при этом x, то текущая кривая видима. В противном случае она невидима.

Полученный результат показан на рис. 5 b.

В изложенном алгоритме предполагается, что значение функции, т.е. y, известно для каждого значения x в пространстве изображения. Однако если для каждого значения x нельзя указать (вычислить) соответствующее ему значение y, то невозможно поддерживать массивы верхнего и нижнего плавающих горизонтов. В таком случае используется линейная интерполяция значений y между известными значениями для того, чтобы заполнить массивы верхнего и нижнего плавающих горизонтов, как показано на рис. 6. Если видимость кривой меняется, то метод с такой простой интерполяцией не даст корректного результата. Этот эффект проиллюстрирован рис. 7 a. Предполагая, что операция по заполнению массивов проводится после проверки видимости, получаем, что при переходе текущей кривой от видимого к невидимому состоянию (сегмент AB на рис. 7 a), точка (X_{N+K}, Y_{N+K}) объявляется невидимой. Тогда участок кривой между точками (X_N, Y_N) и (X_{N+K}, Y_{N+K}) не изображается и операция по заполнению массивов не проводится. Образуется зазор между текущей и предыдущей кривыми. Если на участке текущей кривой происходит переход от невидимого состояния к видимому (сегмент CD на рис. 8 a), то точка (X_{M+K}, Y_{M+K}) объявляется видимой, а участок кривой между точками (X_M, Y_M) и (X_{M+K}, Y_{M+K}) изображается и операция по заполнению массивов проводится. Поэтому изображается и невидимый кусок сегмента CD. Кроме того, массивы плавающих горизонтов не будут содержать точных значений y. А это может повлечь за собой дополнительные нежелательные эффекты для последующих кривых.

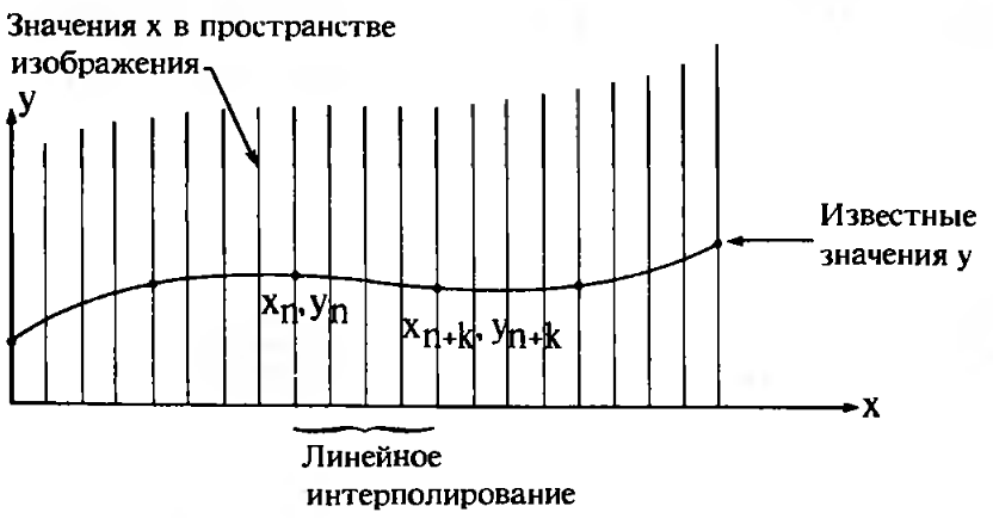

Рисунок 6 – Линейная интерполяция между заданными точками.

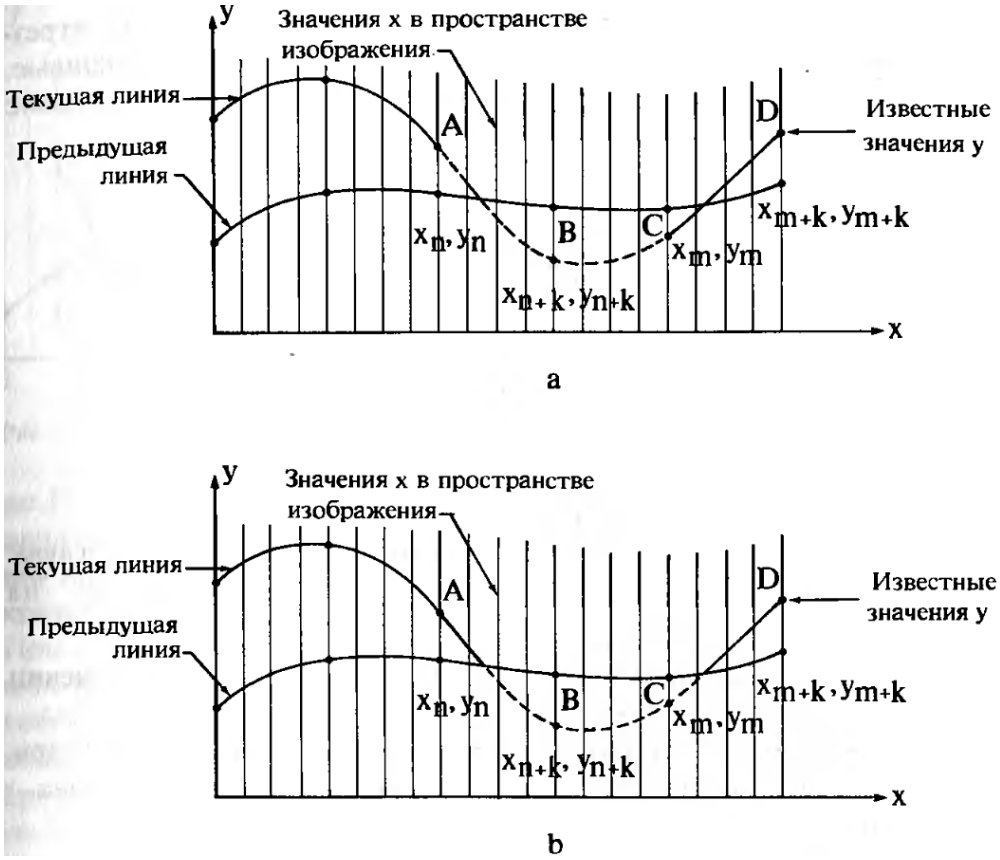

Рисунок 7 – Эффект пересекающихся кривых.

Следовательно, необходимо решать задачу о поиске точек пересечения сегментов текущей и предшествующей кривых.

Существует несколько методов получения точек пересечения кривых. На растровых дисплеях значение координаты x можно увеличивать на 1, начиная с x_n или x_m (рис. 7 а). Значение y, соответствующее текущему значению координаты x в пространстве изображения, получается путем добавления к значению y, соответствующему

предыдущему значению координаты x, вертикального приращения Δy вдоль заданной кривой. Затем определяется видимость новой точки с координатами $(x+1, y+\Delta y)$. Если эта точка видима, то активируется связанный с ней пиксел. Если невидима, то пиксел не активируется, а x увеличивается на 1. Этот процесс продолжается до тех пор, пока не встретится x_{n+k} или x_{m+k}. Пересечения для растровых дисплеев определяются изложенным методом с достаточной точностью.

Теперь алгоритм излагается более формально.

Если на текущей плоскости при некотором заданном значении x соответствующее значение y на кривой больше максимума или меньше минимума по y для всех предыдущих кривых при этом x, то текущая кривая видима. В противном случае она невидима. Если на участке от предыдущего x_n до текущего x_{n+k} значения x видимость кривой изменяется, то вычисляется точка пересечения x_i. Если на участке от x_n до x_{n+k} сегмент кривой полностью видим, то он изображается целиком; если он стал невидимым, то изображается фрагмент от x_n до x_i; если же он стал видимым, то изображается фрагмент от x_i до x_{n+k}. Заполнить массивы верхнего и нижнего плавающих горизонтов.

Типичный результат работы алгоритма можно наблюдать на рисунке 8. [1]

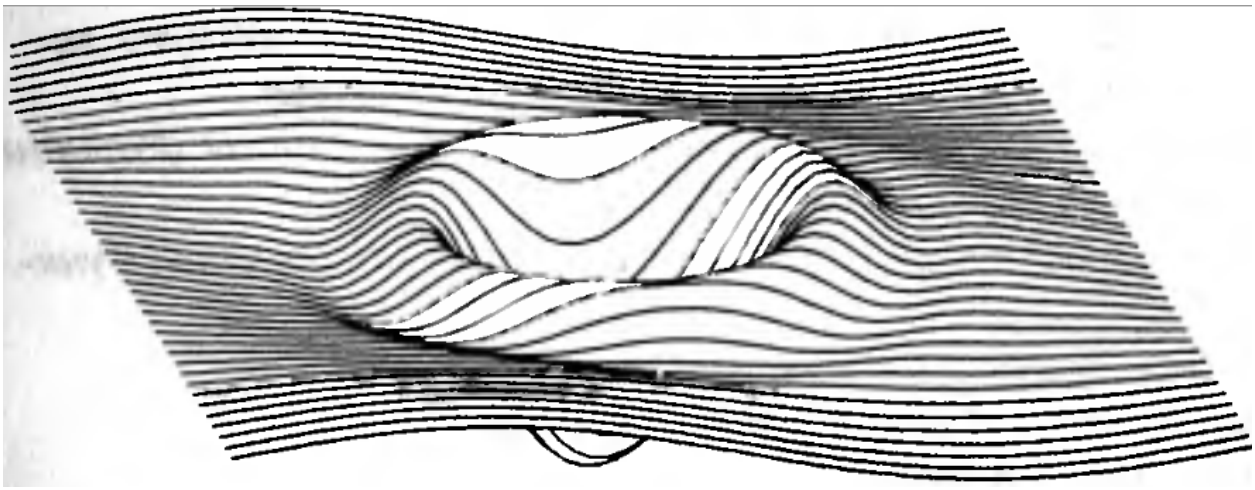

Рисунок 8 – Функция $y = \frac{1}{5}\sin x \cos z - \frac{3}{2}\cos\left(\frac{7a}{4}\right) \times \exp(-a), \quad a = (x-\pi)^2 + (z-\pi)^2$, изображенная в интервале $(0, 2\pi)$ с помощью алгоритма плавающего горизонта.

1.3.3 Алгоритм Робертса

Рассмотрим алгоритм удаления невидимых линий Робертса, требующий, чтобы каждая грань была выпуклым многоугольником.

Сначала отбрасываются все рёбра, обе определяющие грани которых являются нелицевыми (ни одно из таких рёбер заведомо не видно).

Следующим шагом является проверка на закрывание каждого из оставшихся рёбер со всеми лицевыми гранями многогранника. Возможны следующие случаи:

- грань ребра не закрывает (рис. 9);

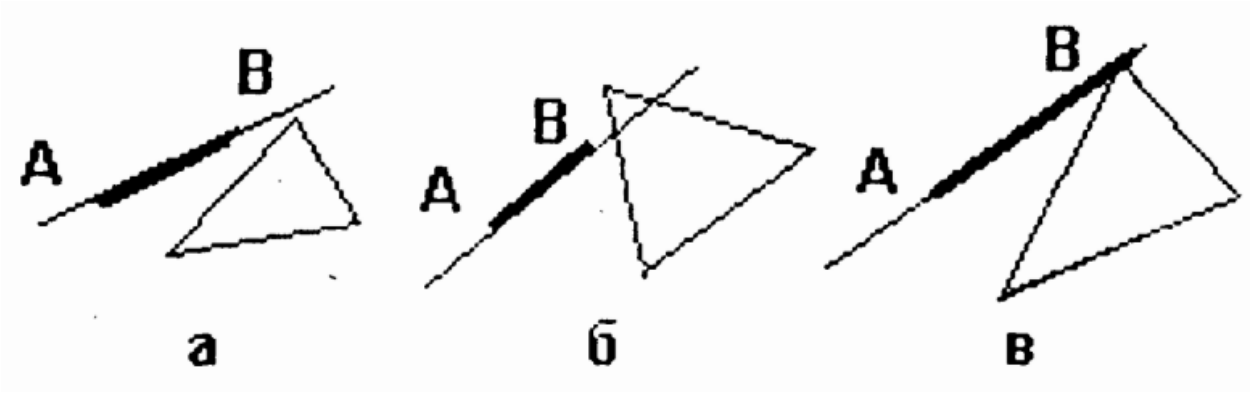

Рисунок 9 – Иллюстрация алгоритма Робертса: (а) грань не закрывает ребро, (б) грань частично закрывает ребро, (в) грань полностью закрывает ребро.

- грань полностью закрывает ребро (тогда оно удаляется из списка рассматриваемых рёбер) (рис. 10, а);
- грань частично закрывает ребро (в этом случае ребро разбивается на несколько частей, видимыми из которых являются не более двух; само ребро удаляется из списка, но в список проверяемых рёбер добавляются те его части, которые данной гранью не закрываются).

Рассмотрим, как осуществляются эти проверки.

Пусть задано ребро AB, где точка A имеет координаты (x_a, y_a), а точка B — (x_b, y_b).

Прямая, проходящая через отрезок AB, задаётся уравнениями

$$x = x_a + t(x_b - x_a), \quad y = y_a + t(y_b - y_a)$$

причем сам отрезок соответствует значениям параметра $0 \le t \le 1$.

Данную прямую можно задать неявным образом как $F(x,y) = 0$, где

$$F(x,y) = (y_b - y_a)(x - x_a) - (y - y_a)(x_b - x_a).$$

Предположим, что проекция грани задается набором проекций вершин $P_1, \ldots, P_k$ с координатами (x_i, y_i), $i = 1, \ldots, n$. Обозначим через F_i значение функции F в точке P_i.

Рассмотрим i-й отрезок проекции грани P_iP_{i+1}. Этот отрезок пересекает прямую AB тогда и только тогда, когда функция F принимает значения разных знаков на концах этого отрезка, а именно при

$$F_iF_{i+1} \le 0.$$

Случай, когда $F_iF_{i+1} = 0$, будем отбрасывать, чтобы дважды не засчитывать прямую, проходящую через вершину, для обоих выходящих из неё отрезков.

Итак, мы считаем, что пересечение имеет место в двух случаях:

$$F_i \ge 0,\ F_{i+1} < 0,$$

$$F_i \le 0,\ F_{i+1} > 0.$$

Точка пересечения определяется соотношениями

$$x = x_i + s(x_{i+1} - x_i),$$

$$y = y_i + s(y_{i+1} - y_i),$$

где $s = \frac{F_i}{F_i - F_{i+1}}$.

Отсюда легко находится значение параметра t:

$$t = \begin{cases} \frac{x - x_a}{x_b - x_a}, & |x_b - x_a| \ge |y_b - y_a|, \\ \frac{y - y_a}{y_b - y_a}, & |y_b - y_a| > |x_b - x_a|. \end{cases}$$

Возможны следующие случаи:

1. Отрезок не имеет пересечений с проекцией грани, кроме, быть может, одной точки. Это может иметь место, когда

   - прямая AB не пересекает рёбра проекции грани (рис. 9, а);
   - прямая AB пересекает ребра в двух точках t_1 и t_2, но либо $t_1 < 0$, $t_2 < 0$, либо $t_1 > 1$, $t_2 > 1$ (рис. 9, б);
   - прямая AB проходит через одну вершину, не задевая внутренности треугольника (рис. 9, в).

   Очевидно, что в этом случае соответствующая грань никак не может закрывать собой ребро AB.

2. Проекция ребра полностью содержится внутри проекции грани (рис. 10, а). Тогда есть две точки пересечения прямой AB и границы грани и $t_1 < 0 < t_2 < 1$. Если грань лежит ближе к картинной плоскости, чем ребро, то ребро полностью невидимо и удаляется.

3. Прямая AB пересекает ребра проекции грани в двух точках и либо $t_1 < 0 \le t_2 \le 1$, либо $0 < t_1 \le 1 < t_2$ (рис. 10, б и в). Если ребро AB находится дальше от картинной плоскости, чем соответствующая грань, то оно разбивается на две части, одна из которых полностью закрывается гранью и потому отбрасывается. Проекция второй части лежит вне проекции грани и поэтому этой гранью не закрывается.

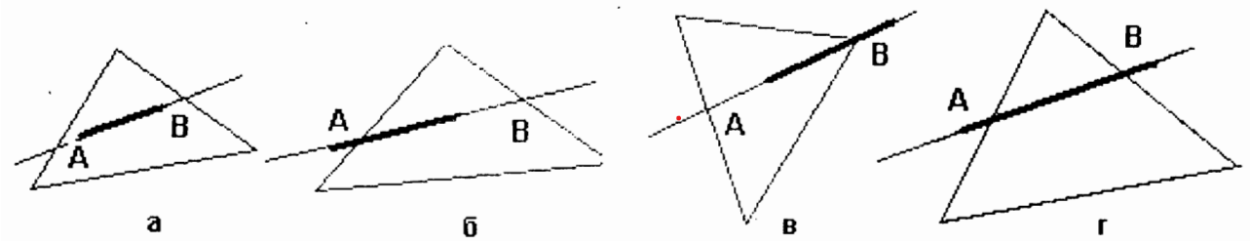

Рисунок 10

4. Прямая AB пересекает ребра проекции грани в двух точках, причем $0 < t_1 < t_2 < 1$ (рис. 10, г). Если ребро AB лежит дальше от картинной плоскости, чем соответствующая грань, то оно разбивается на три части, средняя из которых отбрасывается.

Для определения того, что лежит ближе к картинной плоскости — отрезок AB (проекция которого лежит в проекции грани) или сама грань, через эту грань проводится плоскость

$$(n, p) + c = 0$$

(n — нормальный вектор грани), разбивающая всё пространство на два полупространства. Если оба конца отрезка AB лежат в том же полупространстве, в котором находится и наблюдатель, то отрезок лежит ближе к грани; если оба конца находятся в другом полупространстве, то отрезок лежит дальше. Случай, когда концы лежат в разных полупространствах, здесь невозможен (это означало бы, что отрезок AB пересекает внутреннюю часть грани).

Если общее количество граней равно n, то временные затраты для данного алгоритма составляют $O(n^2)$.

Количество проверок можно заметно сократить, если воспользоваться разбиением картинной плоскости.

Разобьем видимую часть картинной плоскости (экран) на $N_1 \times N_2$ равных частей (клеток) и для каждой клетки A_{ij} построим список всех лицевых граней, чьи проекции имеют с данной клеткой непустое пересечение.

Для проверки произвольного ребра на пересечение с гранями отберем сначала все те клетки, которые проекция данного ребра пересекает. Ясно, что проверять на пересечение с ребром имеет смысл только те грани, которые содержатся в списках этих клеток.

В качестве шага разбиения обычно выбирается $O(l)$, где l — характерный размер ребра в сцене. Для любого ребра количество проверяемых граней практически не зависит от общего числа граней и совокупные временные затраты алгоритма на проверку пересечений составляют $O(n)$, где n — количество ребер в сцене.

Поскольку процесс построения списков заключается в переборе всех граней, их проектировании и определении клеток, в которые попадают проекции, то затраты на составление всех списков также составляют $O(n)$. [4]

1.3.4 Алгоритм Варнака

Джон Варнак, американский учёный, один из основателей компании Adobe Systems. Алгоритм Варнака позволяет определить, какие грани или части граней объектов сцены видимы, а какие заслонены гранями других объектов. Так же как и в алгоритме Робертса анализ видимости происходит в картинной плоскости. В качестве граней обычно выступают выпуклые многоугольники, алгоритмы работы с ними эф-

фективнее, чем с произвольными многоугольниками. Окно, в котором необходимо отобразить сцену, должно быть прямоугольным. Алгоритм работает рекурсивно, на каждом шаге анализируется видимость граней и, если нельзя легко определить видимость, окно делится на 4 части и анализ повторяется отдельно для каждой из частей (см. рис. 11).

Случаи, когда можно легко определить видимость, следующие:

1. в окне нет ни одной грани, в этом случае ничего рисовать не нужно;
2. в окне ровно одна грань, в этом случае достаточно отобразить эту грань, так как нет других граней, которые могли бы её заслонить;
3. размеры окна равны размерам одного пиксела, в этом случае из граней выбирается ближайшая и её цветом закрашивается этот пиксел;
4. ближайшая грань охватывает всё окно, в этом случае она заслоняет все остальные грани и достаточно отобразить только её.

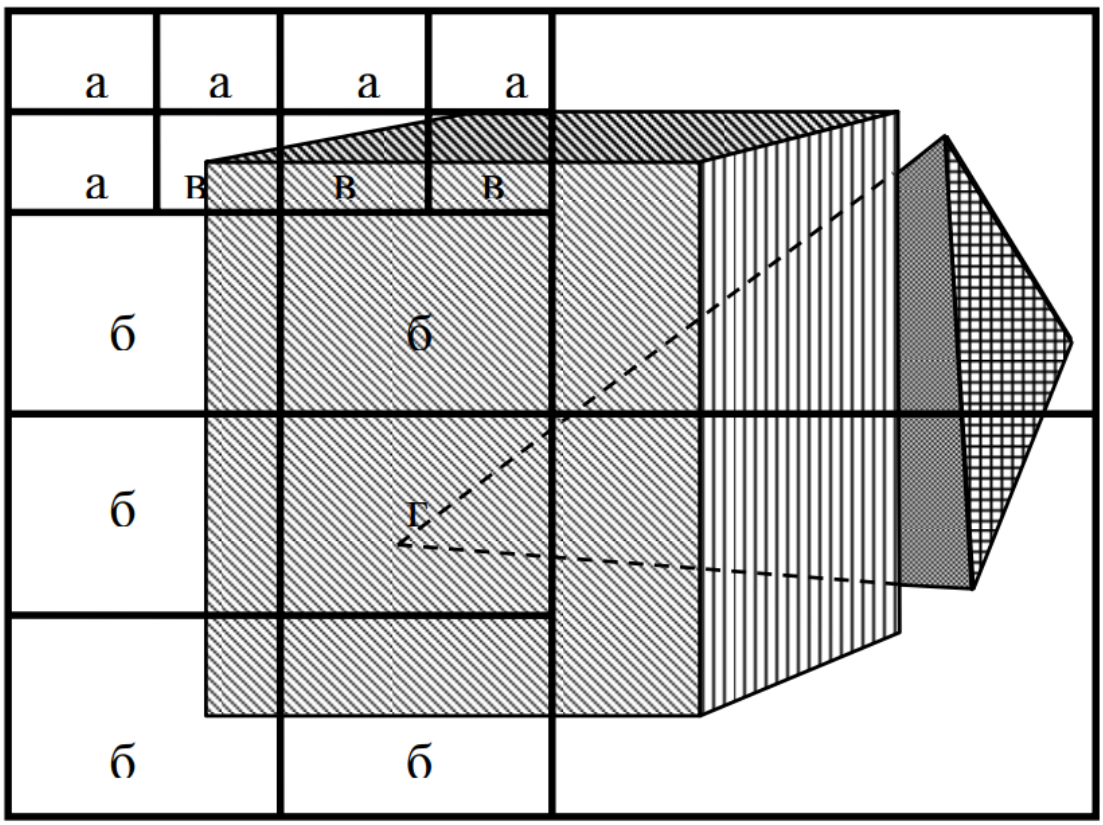

Рисунок 11 – Определение видимости алгоритмом Варнака.

Алгоритм можно разделить на следующие элементарные задачи:

- определение, в какие части окна попадает грань;
- определение, что грань охватывает окно;
- определение, что грань является ближайшей.

Существует два различных подхода к решению этих задач. Сначала рассмотрим вариант решения первой задачи, когда при разделении окна грани разрезаются на части вертикальной и горизонтальной прямыми, делящими окно. Получившиеся многоугольники полностью лежат в соответствующих частях окна, и заносятся в соответствующий этой части набор. При последующем делении окна они разрезаются на более мелкие части. Определить, что такой многоугольник охватывает всё окно можно по условию, что каждая из четырёх вершин окна совпадает с одной из вершин многоугольника. Определить, что грань ближайшая можно по глубине вершин многоугольника. При перспективной проекции в качестве параметра глубины удобно использовать обратную величину $w = 1/z$, где z – координата глубины, так как w в отличие от z можно линейно интерполировать в экранных координатах. Чем больше w, тем ближе вершина к наблюдателю. Так как грань не обязательно параллельна экрану, то глубина у вершин разная. Найдя минимум и максимум глубины всех вершин можно определить интервал глубины [Wmin, Wmax]. В случае, когда окно равно пикселу, достаточно выбрать грань с максимальным Wmax, она и будет ближайшей. Если же нужно убедиться, что грань ближе всех остальных граней, то необходимо проверить условие, что Wmin этой грани больше Wmax всех остальных граней.

Второй подход не требует деления граней на части, для вычислений всегда используются исходные грани.

Для решения первой задачи нужно определить факт пересечения двух выпуклых многоугольников – грани и окна. Это можно сделать, например, по следующему условию: многоугольники не пересекаются, если все вершины одного из них лежат с внешней стороны от прямой, образующей ребро другого (см. рис. 12). Нужно перебрать все рёбра грани и сравнить с вершинами окна, и все четыре границы окна сравнить с вершинами грани.

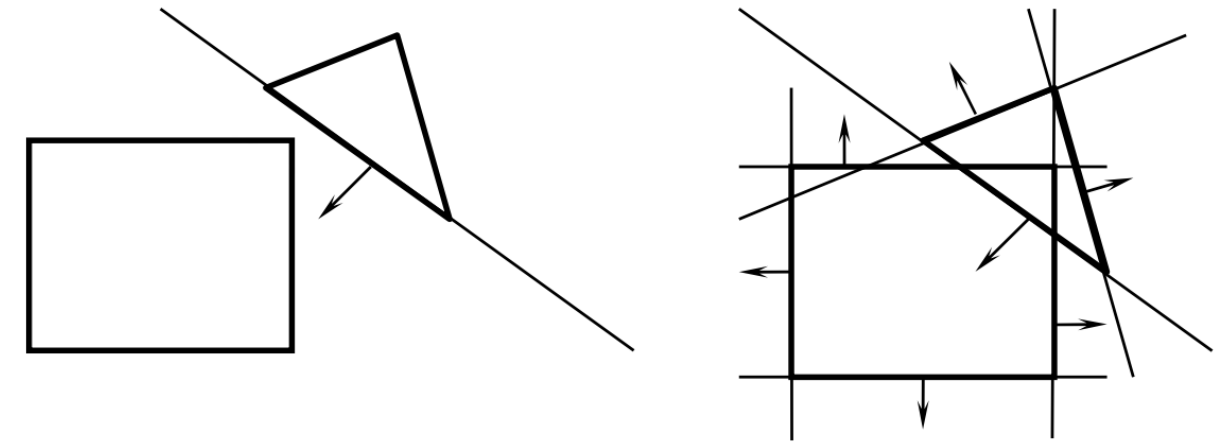

Рисунок 12 – Определение пересечения выпуклых многоугольников.

Для решения второй задачи достаточно проверить, что все 4 угла окна лежат

внутри или на границе грани. Это можно определить по расстояниям от углов окна до прямых, образующих рёбра граней.

Третья задача решается сложнее, чем в первом варианте. Грань не лежит полностью в окне, поэтому рассчитывать интервал её глубины по вершинам будет неправильно. Необходимы вершины той её части, которая лежит в окне. Это вершины грани, попадающие в окно, вершины окна, попадающие в грань и точки пересечения рёбер грани с рёбрами окна (см. рис. 13).

По этим точкам рассчитывается интервал глубины [Wmin, Wmax] и используется для определения ближайшей грани.

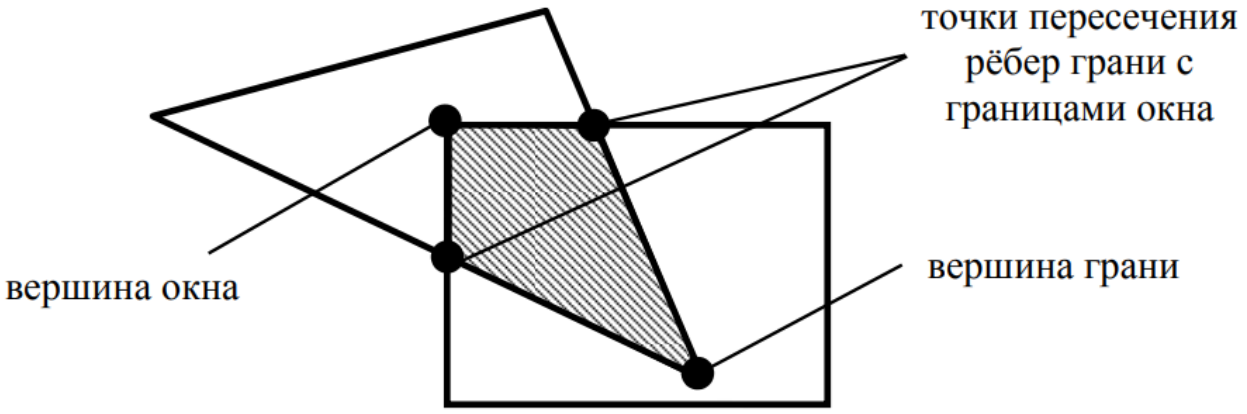

Рисунок 13 – Определение вершин пересечения грани с окном.

У каждого из подходов есть свои достоинства и недостатки. Подход с разрезанием граней проще в реализации, но требует дополнительной памяти для хранения частей граней. Второй подход лишён этого недостатка, но более сложен в реализации. Производительность при обоих подходах примерно одинакова. Её можно оценить, исходя из количества граней N и количества пикселей C как O(CN), так как в худшем случае все грани могут быть размером с окно и окно в худшем случае придётся делить до пиксела. Уменьшить количество граней нельзя, но можно сократить количество делений окна. Например, его можно сократить, если дополнительно обрабатывать случай на общем ребре двух граней. По описанному алгоритму окно, в которое попадает общее ребро двух граней, будет делиться до пиксела. Это происходит потому, что не выполняется ни один из четырёх случаев, в окне несколько граней и ни одна из этих граней не охватывает окно (см. рис. 14).

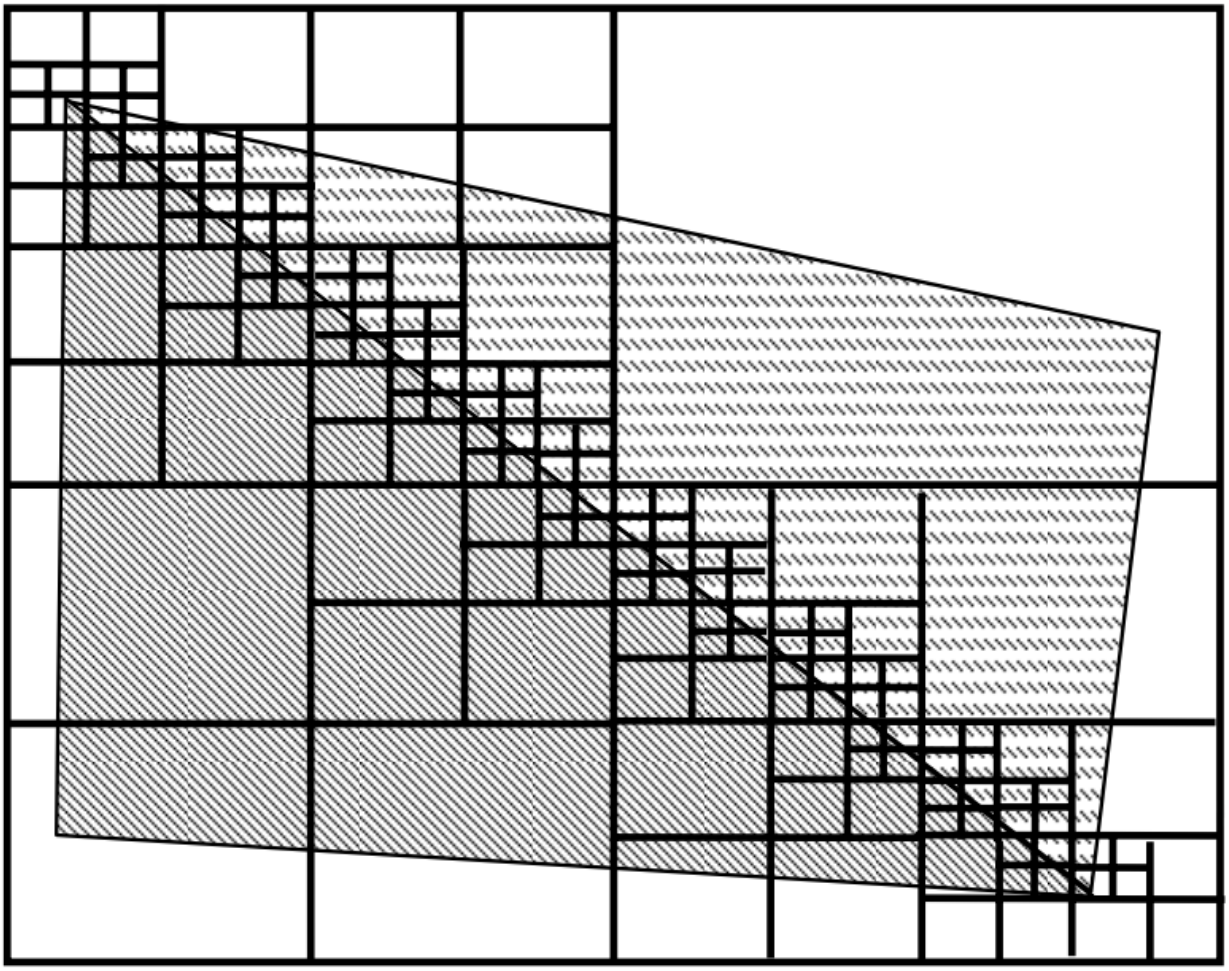

Рисунок 14 – Разделение окна вдоль общего ребра двух граней.

Можно избежать лишних делений, если выделить общие рёбра граней, и добавить ещё один простой случай: две грани с общим ребром полностью охватывают окно и являются ближайшими. Ребро в этом случае должно пересекать окно в двух точках, вершины окна с одной стороны ребра должны лежать внутри грани лежащей с этой стороны ребра, а вершины окна с другой стороны ребра должны лежать внутри второй грани (см. рис. 15).

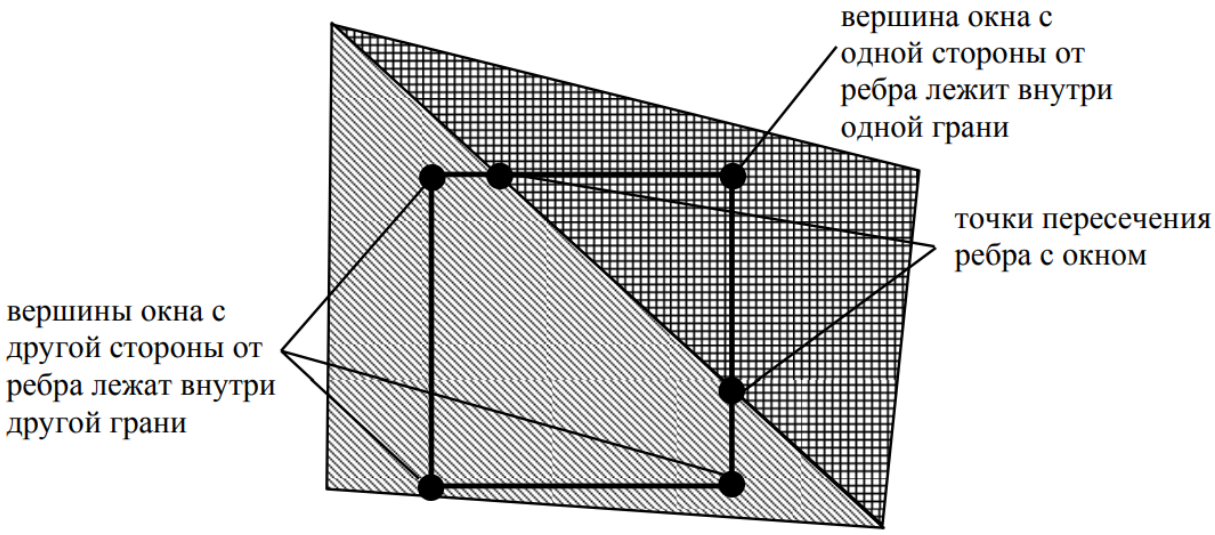

Рисунок 15 – Определение случая с двумя гранями с общим ребром.

Интервал глубины [Wmin, Wmax] должен рассчитываться по вершинам обеих граней, попавших в окно частей. [3]

1.3.5 Алгоритм Вейлера-Азертона

Вейлер и Азертон попытались минимизировать количество шагов в алгоритме разбиения типа алгоритма Варнока путем разбиения окна вдоль границ многоугольника. Выходными данными этого алгоритма, который для достижения необходимой точности работает в пространстве объекта, служат многоугольники. Поскольку выходом являются многоугольники, то алгоритм можно легко использовать для удаления как невидимых линий, так и невидимых поверхностей. Алгоритм удаления невидимых поверхностей состоит из четырех шагов.

1. Предварительная сортировка по глубине.
2. Отсечение по границе ближайшего к наблюдателю многоугольника, называемое сортировкой многоугольников на плоскости.
3. Удаление многоугольников, экранированных многоугольником, ближайшим к точке наблюдения.
4. Если требуется, то рекурсивное подразделение и окончательная сортировка для устранения всех неопределенностей.

Предварительная сортировка по глубине нужна для формирования списка приблизительных приоритетов. Предположим, что точка наблюдения расположена в бесконечности на положительной полуоси z, тогда ближайшим к ней и первым в списке будет тот многоугольник, который обладает вершиной с максимальной координатой z.

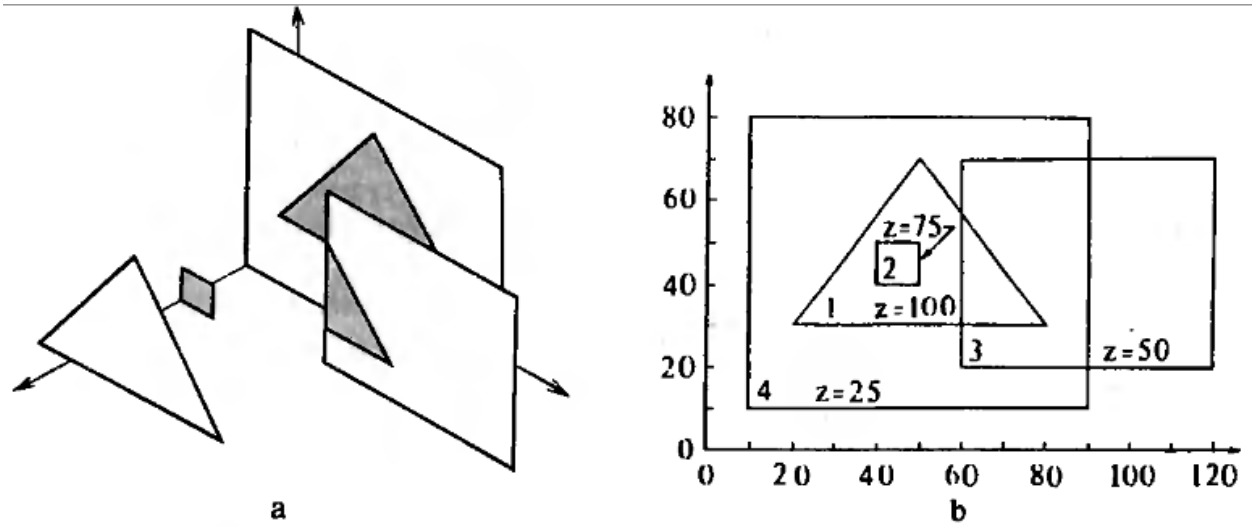

Рисунок 16 – Отсечение многоугольников по приоритетам для алгоритма удаления невидимых поверхностей Вейлера — Азертона.

В качестве отсекающего многоугольника используется копия первого многоугольника из предварительного списка приоритетов по глубине. Отсекаться будут оставшиеся в этом списке многоугольники, включая и первый многоугольник. Вводятся два списка: внутренний и внешний. С помощью алгоритма отсечения Вейлера — Азертона все многоугольники отсекаются по границам отсекающего многоугольника. Фактически это двумерная операция отсечения проекций отсекающего и отсекаемого многоугольников. Та часть каждого отсекаемого многоугольника, которая оказывается внутри отсекающего, если она имеется, попадает во внутренний список. Оставшаяся часть, если таковая есть, попадает во внешний список. Этот этап алгоритма является сортировкой на плоскости или xy-сортировкой. Пример приведен на рис. 16. На рис. 17 показаны внутренний и внешний списки для сцены на рис. 16. Теперь сравниваются глубины каждого многоугольника из внутреннего списка с глубиной отсекающего многоугольника. С использованием координат (x,y) вершин отсекаемых многоугольников и уравнений несущих плоскостей вычисляются глубины (координаты z) каждой вершины. Затем они сравниваются с минимальной координатой $z(z_{\min})$ для отсекающего многоугольника. Если глубина ни одной из этих вершин не будет больше $z_{\min}$, то все многоугольники из внутреннего списка экранируются отсекающим многоугольником (рис. 16). Эти многоугольники удаляются, и выбирается следующий многоугольник. Заметим, что во внутренний список остается лишь отсекающий многоугольник. Работа алгоритма затем продолжается с внешним списком.

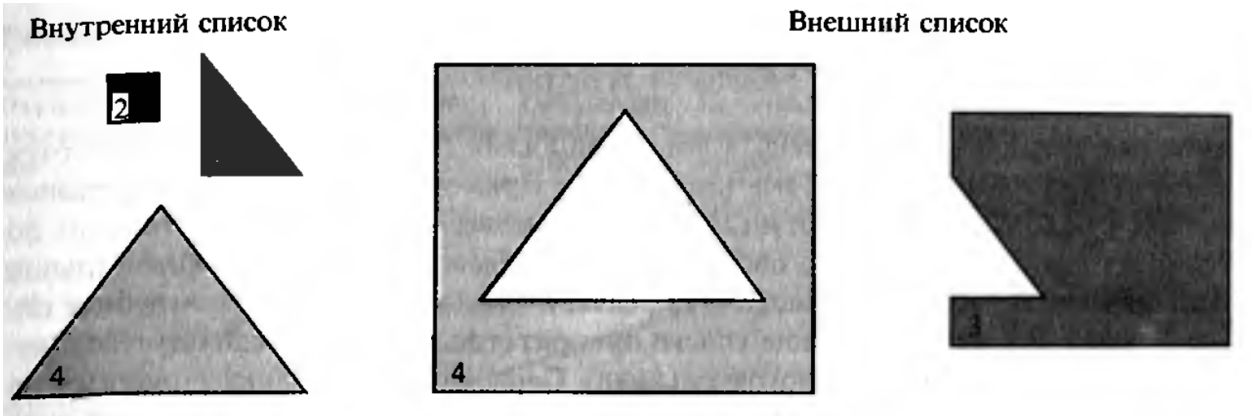

Рисунок 17 – Внутренний и внешний списки многоугольников.

Если координата z какого-либо многоугольника из внутреннего списка окажется больше, чем $z_{\text{c min}}$, то такой многоугольник по крайней мере частично экранирует отсекающий многоугольник. На рис. 18 показано, как это может произойти. В подобном случае результат предварительной сортировки по глубине ошибочен. Поэтому алгоритм рекурсивно подразделяет плоскость (x,y), используя многоугольник, нарушивший порядок, в качестве нового отсекающего многоугольника. Отсечению подлежат многоугольники из внутреннего списка, причем старый отсекающий многоугольник теперь сам будет подвергнут отсечению новым отсекающим многоугольником. Подчеркнем, что новый отсекающий многоугольник является копией исходного многоугольника, а не его остатка после первого отсечения. Использование копии неотсекаемого многоугольника позволяет минимизировать число разбиений.

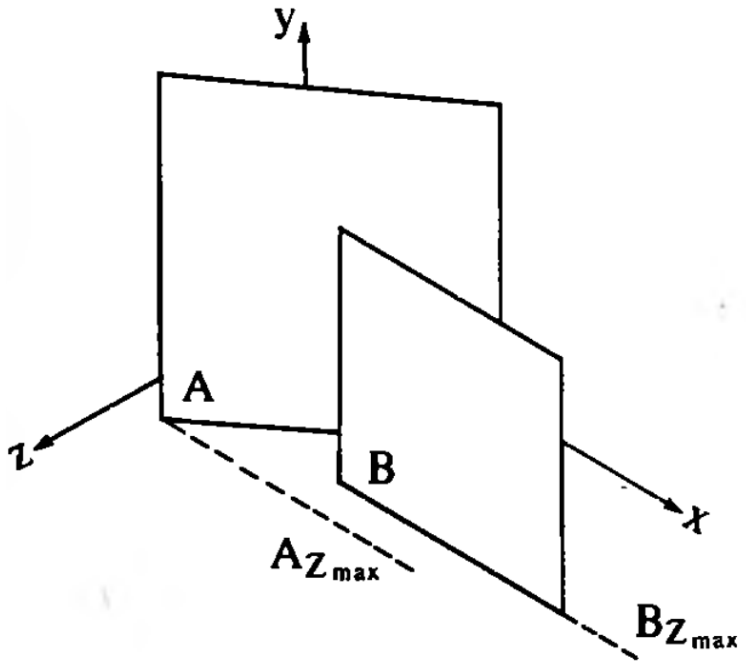

Рисунок 18 – Условие возникновения ошибочного результата в предвари- тельной сортировке по z.

Заслуживает внимание еще одна дополнительная деталь этого алгоритма. Ко-

гда некоторый многоугольник циклически перекрывается с отсекающим, т. е. когда он лежит как впереди, так и позади отсекающего (рис. 19, a), то в рекурсоном разбиении необходимости нет. дело в том, что все экранируемое циклическим многоугольником уже было удалено на предыдущем шаге отсечения. Необходимо лишь произвести отсечение исходного многоугольника по границам циклического многоугольника и изобразить результат. Ненужное рекурсивное разбиение можно предотвратить, если создать список многоугольников, которые уже использовались как отсекающие. Тогда, если при рекурсивном разбиении текущий отсекающий многоугольник появляется в этом списке, значит, обнаружен циклически перекрывающийся многоугольник. Следовательно, не требуется никакого дополнительного разбиения. Заметим, что данный алгоритм непосредственно обрабатывает случаи циклического перекрытия сразу нескольких многоугольников, как показано на рис. 19, b. [1]

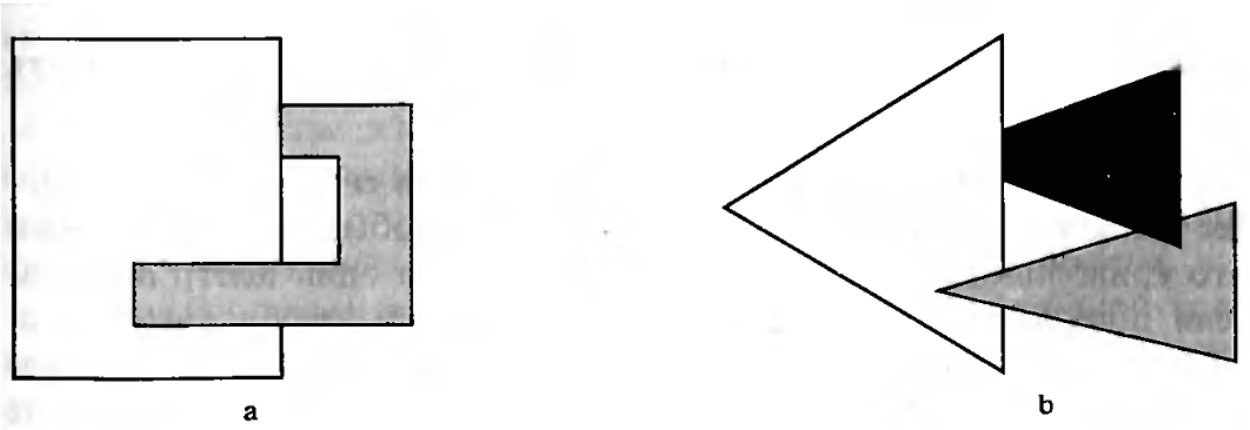

Рисунок 19 – Циклически перекрывающиеся многоугольники.

1.3.6 Алгоритм художника

Алгоритм художника позволяет определить, какие пикселы граней сцены видимы, а какие заслонены гранями других объектов. Для этого все грани сортируются и рисуются в порядке от дальних к ближним, в результате чего ближние грани автоматически заслоняют дальние (см. рис. 20).

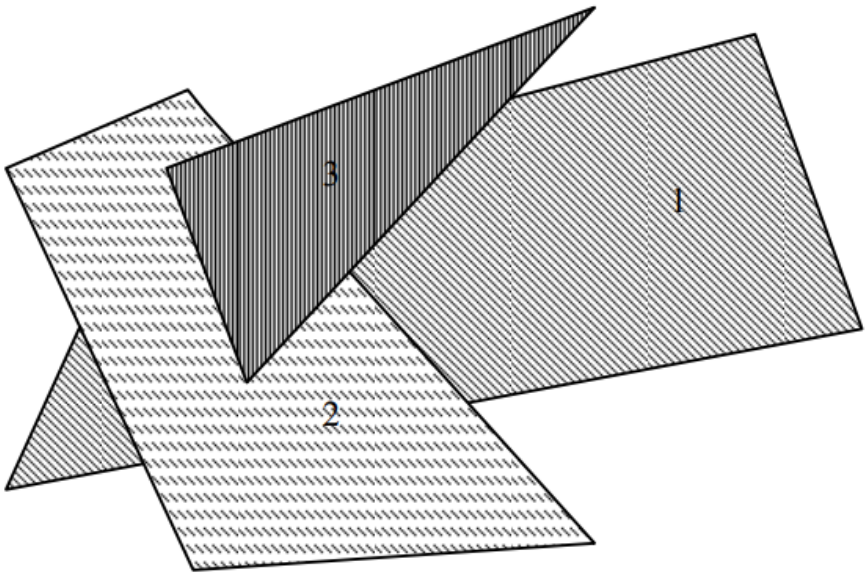

Рисунок 20 – Отрисовка граней по алгоритму художника.

Сортировка граней от дальних к ближним задача нетривиальная, недостаточно отсортировать грани по координате, соответствующей глубине. Каждой грани обычно соответствует некоторый интервал глубины, и если интервалы двух граней пересекаются, то нужны дополнительные проверки, чтобы определить, какая грань ближе. Кроме того, встречаются ситуации, когда грани отсортировать невозможно, например, когда грани образуют цикл (см. рис. 21а). В этом случае нужно разделить одну из граней на части (см. рис. 21б), и тогда становится возможным определить порядок рисования.

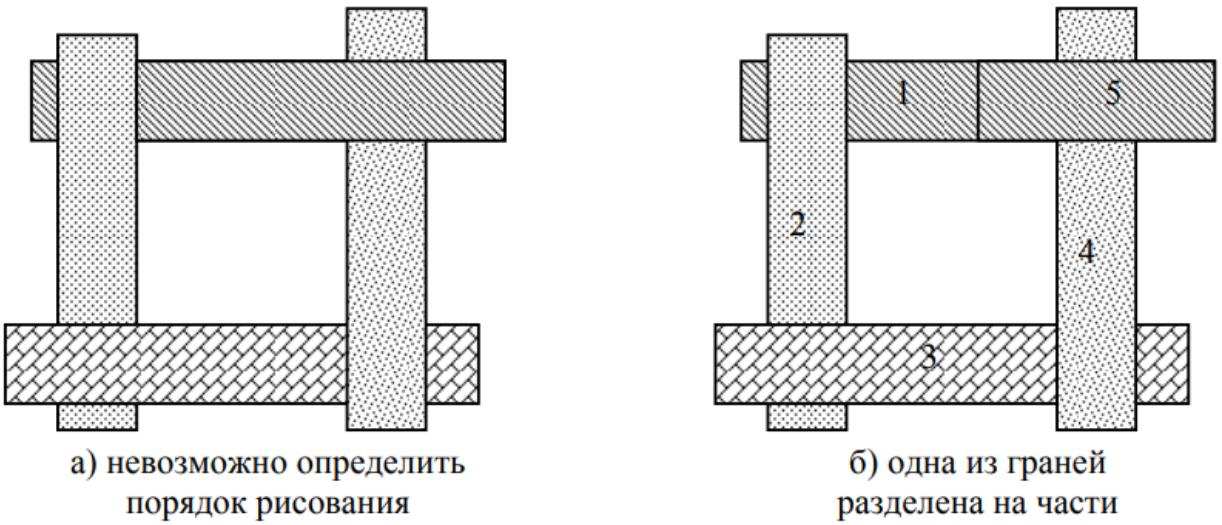

Рисунок 21 – Циклически закрывающие друг друга грани.

Одним из методов сортировки граней является метод бинарного разделения пространства. В качестве исходных данных выступает набор граней в пространстве и точка, в которой находится наблюдатель. Если разделить пространство плоскостью одной из граней, то грани, лежащие в полупространстве, в котором находится наблюдатель, будут ближе к нему, чем те, что лежат в другом полупространстве. Поэтому порядок

рисования граней будет следующим:

1. грани, лежащие в полупространстве, где нет наблюдателя;
2. грани, лежащие на разделяющей плоскости;
3. грани, лежащие в полупространстве, где находится наблюдатель.

Аналогично сортируются и грани в полупространствах, в каждом выбирается грань, её плоскостью полупространство делится на две части и так до тех пор, пока в каждой части не останется ни одной грани. В качестве примера рассмотрим случай на плоскости (см. рис. 22), вместо граней – отрезки (a, b, c, d, e, f, g, h), вместо плоскостей – прямые (можно представить, что все грани вертикальные, а на рисунке показан их вид сверху).

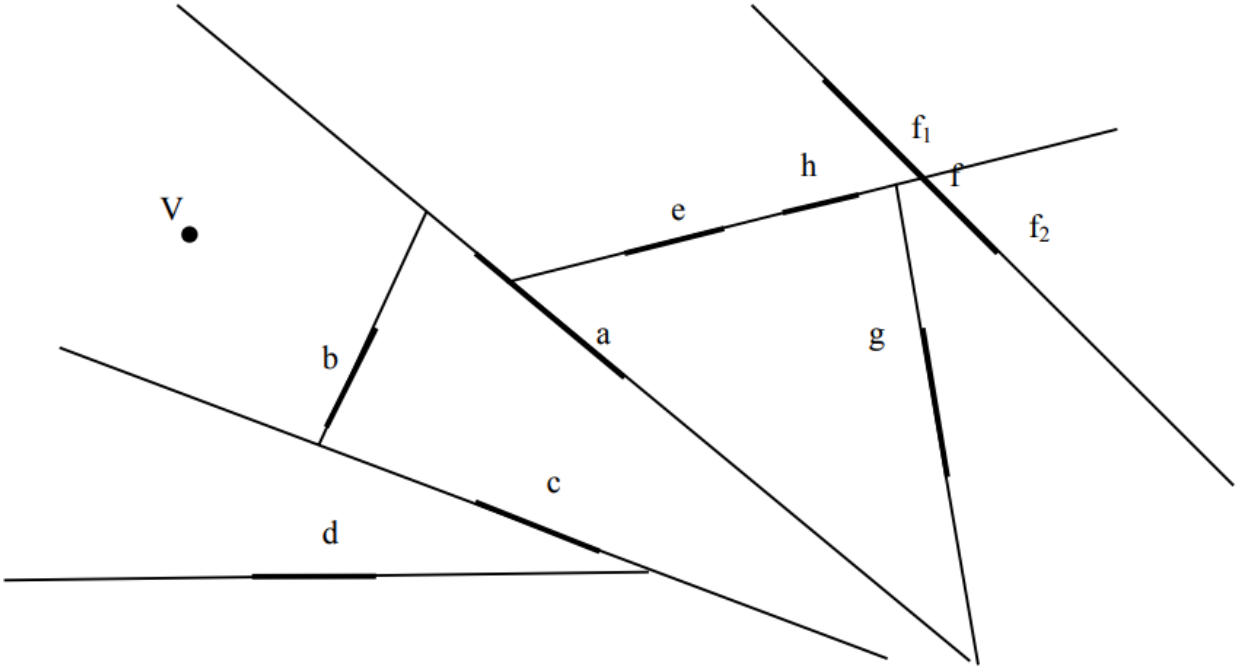

Рисунок 22 – Сортировка граней бинарным разделением пространства.

Наблюдатель находится в точке V. Плоскость первой грани (a) делит грани на три группы:

- (b,c,d) – лежат в полупространстве наблюдателя,
- (e,g,h,f) – в другом полупространстве,
- грань (a) – на разделяющей плоскости.

Порядок рисования: (e,g,h,f), (a), (b,c,d). Грани в каждом из полупространств нужно отсортировать между собой. В одном выбирается плоскость грани (e), она делит грани на группы:

- (f_1) – лежит в полупространстве наблюдателя,
- (f_2, g) – в другом полупространстве,
- грани (e, h) – на разделяющей плоскости.

Грань (f) попала в оба полупространства и была разделена на две части (f_1 и f_2), грань (h) попала на разделяющую плоскость и должна рисоваться вместе с гранью (e). Их взаимный порядок рисования может быть любым, так как они не могут заслонить друг друга, находясь в одной плоскости.

Аналогично, в части пространства с гранями (f_2, g) выбирается плоскость g, а в части пространства с гранями (b, c, d) – грань (c). После всех разделений порядок рисования граней будет:

$$f_2, g, e, h, f_1, a, d, c, b.$$

Время, необходимое на сортировку, зависит от количества граней на всех этапах разделения пространства. Например, при первом делении нужно сравнить с разделяющей плоскостью все N граней. Количество граней может возрасти из-за разделения граней на части, поэтому лучше выбирать такие плоскости, которые делят минимум граней. Также алгоритм будет работать эффективнее, если с каждой стороны от разделяющей плоскости будет примерно одинаковое количество граней.

Рассмотрим два крайних случая:

1. Все грани оказываются с одной стороны от плоскости, кроме одной, которая является разделяющей. Тогда количество сравнений на первом шаге будет N, на втором $N-1$, на третьем $N-2$ и так далее до 1. Общее количество делений будет N, а сложность $O(N^2)$.

2. Грани с обеих сторон распределены поровну. Количество сравнений на первом шаге будет N, на втором $N/2+N/2$ (по $N/2$ с каждой стороны), на третьем $N/4+N/4+N/4+N/4$ и так далее, пока не останется $1+1+\ldots+1$. Общее количество делений будет $\log N$, а сложность $O(N \cdot \log N)$. [3]

1.3.7 Алгоритм трассировки лучей

Алгоритм трассировки лучей позволяет определить, какие пикселы граней сцены видимы, а какие заслонены гранями других объектов. Определение происходит непосредственно для каждого пиксела. Для этого ищется пересечение луча, исходящего из точки наблюдателя через центр пиксела, с гранями сцены и выбирается ближайшая к наблюдателю грань, она и будет видима в данном пикселе (см. рис. 23).

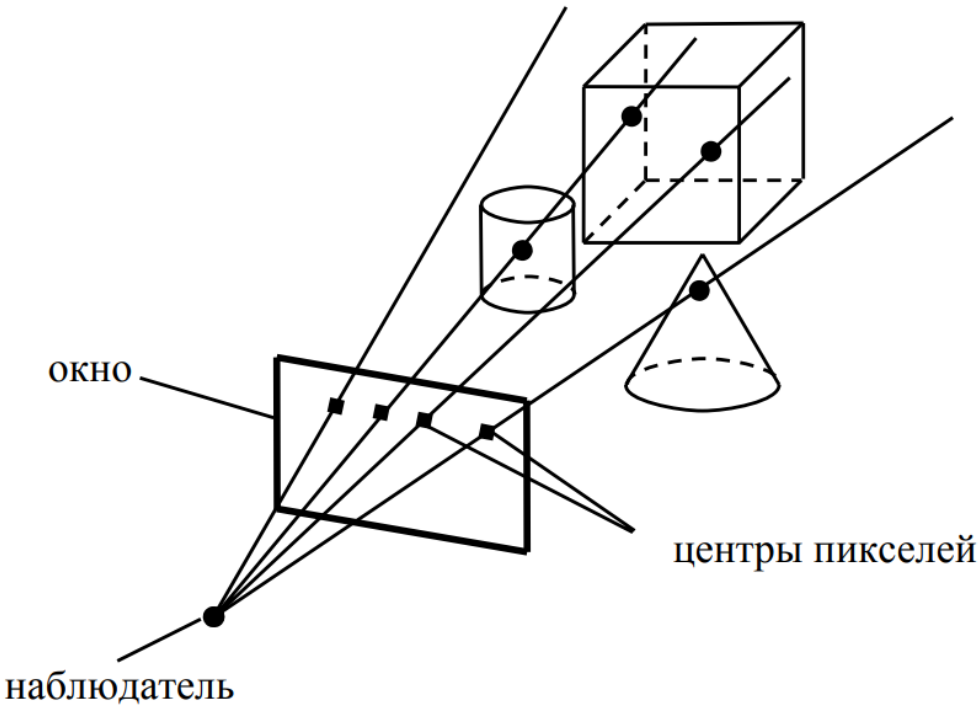

Рисунок 23 – Трассировка лучей.

Скорость работы алгоритма можно оценить как O(CN), где N – количество граней, C – количество пикселей, так как необходимо луч для каждого пиксела проверить на пересечение с каждой из граней. Для повышения производительности можно воспользоваться когерентностью в пространстве. Грани обычно образуют поверхность объектов, то есть располагаются в пространстве компактными группами. Вокруг таких групп можно описать простые оболочки, например, сферы, и сначала проверять луч на пересечение с оболочкой, и только если он с ней пересекается, искать пересечение с отдельными гранями. Также можно использовать когерентность в картинной плоскости. Если для текущего пиксела была найдена ближайшая грань, то она скорее всего будет ближайшей и для соседних пикселей. Таким образом, для соседнего пиксела можно сначала проверить луч на пересечение с этой гранью. Если оно имеет место, то с остальными гранями достаточно проверять пересечение не всего луча, а только отрезка от наблюдателя до найденной точки пересечения. Это позволит быстрее отбросить грани или описанные тела, находящиеся дальше от наблюдателя, чем эта точка пересечения. [3]

1.3.8 Алгоритм Z-буфера

Алгоритм, использующий z-буфер это 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 пиксела в пространстве изображения, z-буфер - это отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пиксела

в пространстве изображения. В процессе работы глубина или значение z каждого нового пиксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксел 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 512´512´24 бит в комбинации с z-буфером размером 512´512´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.

Альтернативой созданию специальной памяти для z-буфера является использование для этой цели оперативной памяти. Уменьшение требуемой памяти достигается разбиением пространства изображения на 4, 16 или больше квадратов или полос. В предельном варианте можно использовать z-буфер размером в одну строку развертки. Для последнего случая имеется интересный алгоритм построчного сканирования. Поскольку каждый элемент сцены обрабатывается много раз, то сегментирование z-буфера, вообще говоря, приводит к увеличению времени, необходимого для обработки сцены. Однако сортировка на плоскости, позволяющая не обрабатывать все многоугольники в каждом из квадратов или полос, может значительно сократить этот рост.

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

некорректном порядке, что ведет к локальным ошибкам.

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 Пиксел(x,y) в многоугольнике вычислить его глубину z(x,y).
5. Сравнить глубину z(x,y) со значением Zбуфер(x,y), хранящимся в z-буфере в этой же позиции.

Если z(x,y) $>$ Zбуфер (x,y), то записать атрибут этого многоугольника (интенсивность, цвет и т.п.) в буфер кадра и заменить Zбуфер(x,y) на z(x,y). В противном случае никаких действий не производить. [6]

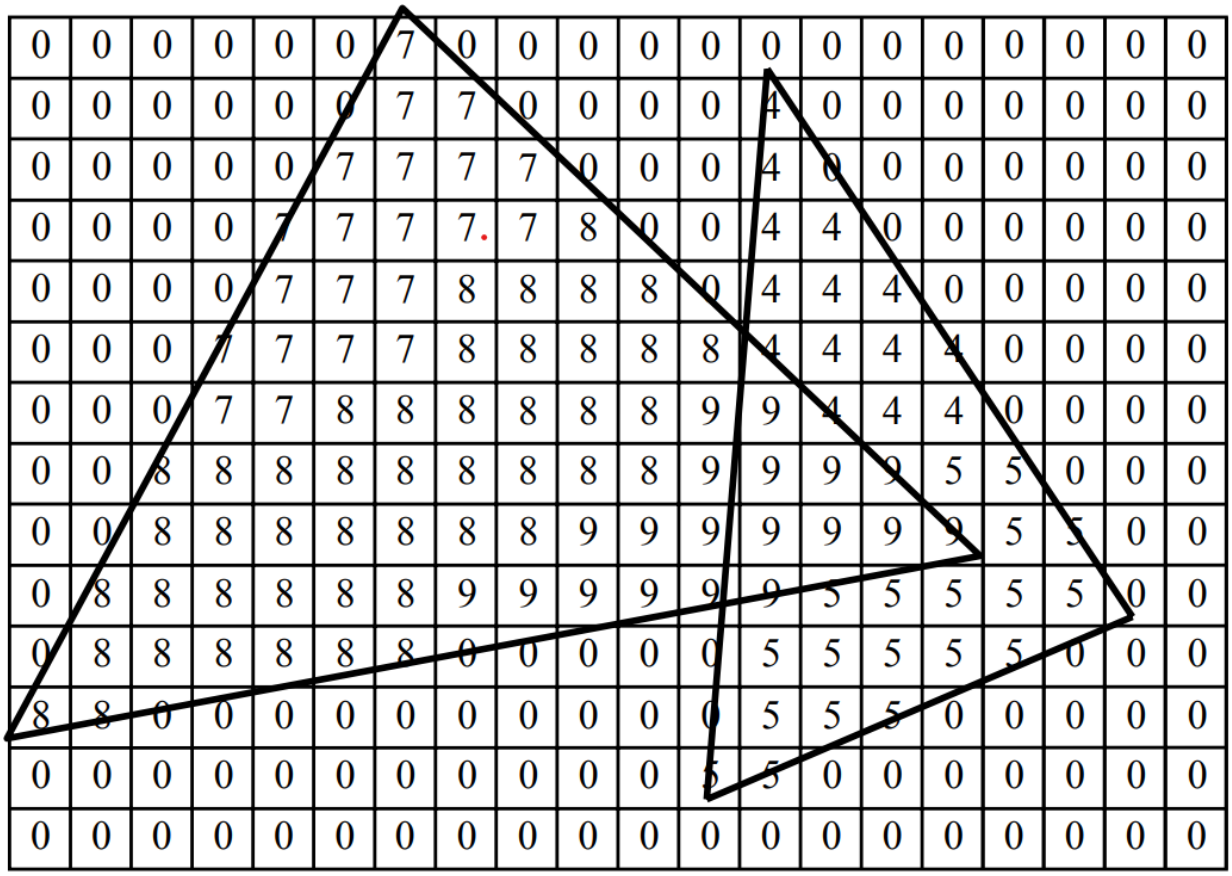

Рисунок 24 – Значения в Z-буфере.

1.3.9 Методы построчного сканирования

Суть метода построчного сканирования заключается в том, что через каждую строку пикселей окна проводится плоскость, дающая сечение сцены. При пересечении

поверхностей объектов этими плоскостями образуются некоторые линии, например, при пересечении граней – отрезки, и задача видимости решается для этого набора отрезков (см. рис. 25). Таким образом, задача определения видимых участков граней в трёхмерном пространстве преобразуется в набор задач определения видимости отрезков на плоскости, которые решаются проще и эффективнее.

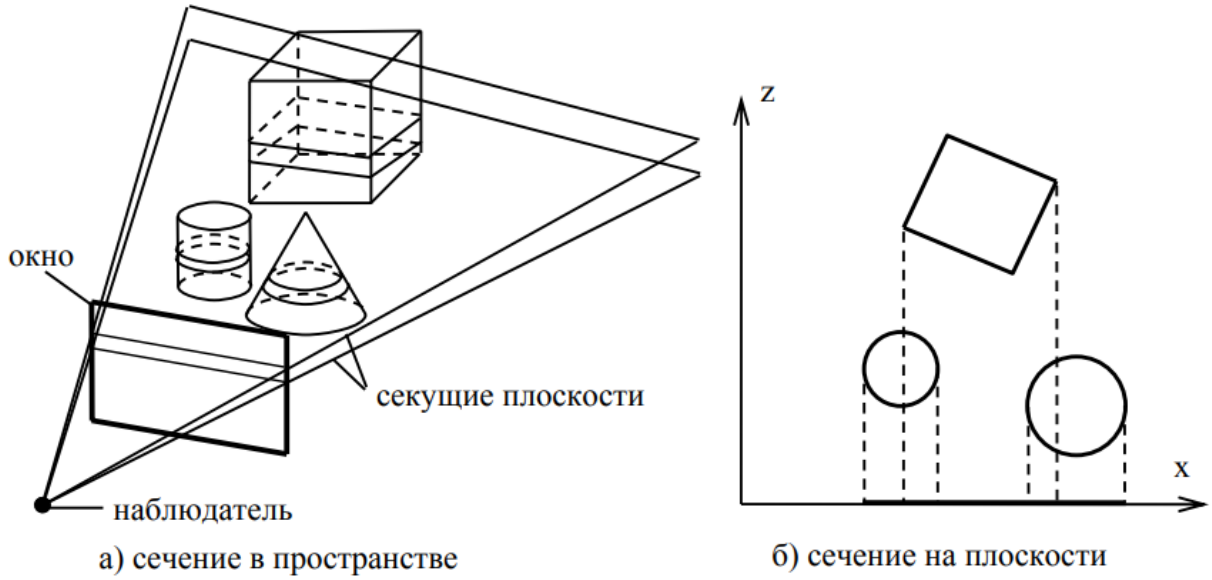

Рисунок 25 – Метод построчного сканирования.

Пересечение плоскости и выпуклой грани является отрезком, на экране он отображается в виде горизонтального отрезка пикселей – спана. [3]

1.3.9.1 Алгоритм построчного Z-буффера

Алгоритм работает аналогично обычному Z-буферу (см. разд. 1.3.8), единственное отличие в том, что вместо двухмерного массива значений глубины, достаточно одномерного массива размером в строку пикселей. Каждому пикселу соответствует свой элемент массива, перед рисованием строки в него заносится максимальное значение глубины, а при рисовании пикселей спанов их глубина сравнивается с глубиной в Z-буфере и, если пиксел ближе, он рисуется, а его глубина заносится в Z-буфер. Построчный алгоритм более эффективен, чем полноэкранный, так как Z-буфер требует значительно меньшего объёма памяти и полностью умещается в кэше процессора. Чтобы ещё немного повысить производительность можно воспользоваться когерентностью соседних строк, грани видимые в одной строке, скорее всего, видимы и в соседних. Если рисовать сначала те грани, которые были видимы в предыдущей строке, а затем все остальные, то Z-буфер будет сразу заполняться наиболее близкими значениями глубины этих граней, а большинство пикселей остальных граней окажутся дальше, что позволит сократить количество операций рисования пикселей и модификации значений глубины в Z-буфере. [3]

1.3.9.2 Алгоритм S-буффера

S-буфер – это буфер отрезков видимых наблюдателю (S – от segment, отрезок). Отрезки, полученные в результате пересечения граней плоскостью, поочерёдно заносятся в S-буфер. При этом очередной отрезок сначала сравнивается с отрезками буфера. Могут возникнуть следующие случаи:

- проекции отрезков не пересекаются (см. рис. 26а), ни один из отрезков не заслоняет другого;
- новый отрезок может полностью заслонить отрезок буфера (см. рис. 26б), в этом случае заслонённый отрезок удалятся из буфера;
- новый отрезок может заслонить часть отрезка буфера (см. рис. 26в), в этом случае отрезок буфера делится на части и заслонённая часть удаляется из буфера;
- отрезок из буфера может полностью заслонить новый отрезок (см. рис. 26б), в этом случае новый отрезок не виден и не заносится в буфер;
- отрезок из буфера может заслонить часть нового отрезка (см. рис. 26в), в этом случае новый отрезок делится на части, заслонённая часть удаляется, а оставшиеся сравниваются с остальными отрезками буфера;
- отрезки пересекаются (см. рис. 26г), в этом случае оба отрезка делятся на части, заслонённые части удаляются, оставшиеся части отрезка буфера заносятся обратно в буфер, оставшиеся части нового отрезка сравниваются с остальными отрезками буфера.

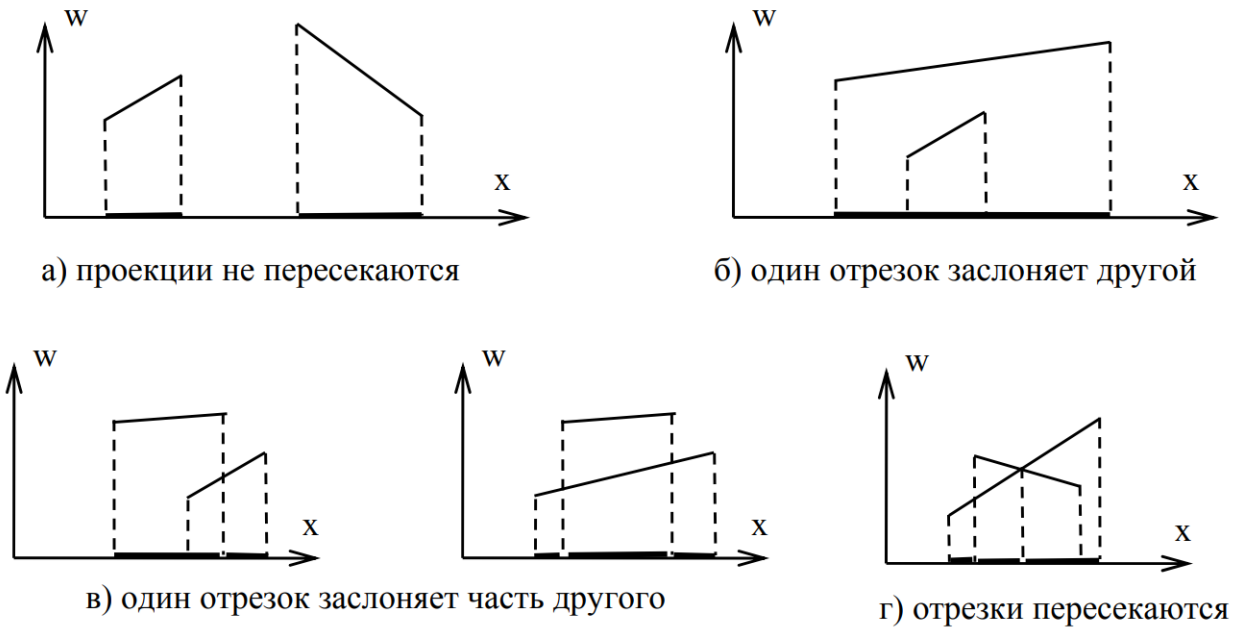

Рисунок 26 – Сравнение двух отрезков.

На рисунке 26 ось x – горизонтальная ось окна, ось w – параметр глубины, чем больше w, тем точка ближе к наблюдателю.

Определить, что проекции отрезков не пересекаются можно по условию, что максимальная координата x одного отрезка не превышает минимальную координату x другого. Когда проекции пересекаются, нужно определить, какие части отрезков заслонены (см. рис. 27). Если отсортировать точки концов отрезков (A, B, C, D) вдоль оси x, то можно выделить три интервала (A-B, B-C, C-D). На первом интервале видна часть отрезка, которому принадлежит точка A, на третьем интервале видна часть отрезка, которому принадлежит точка D.

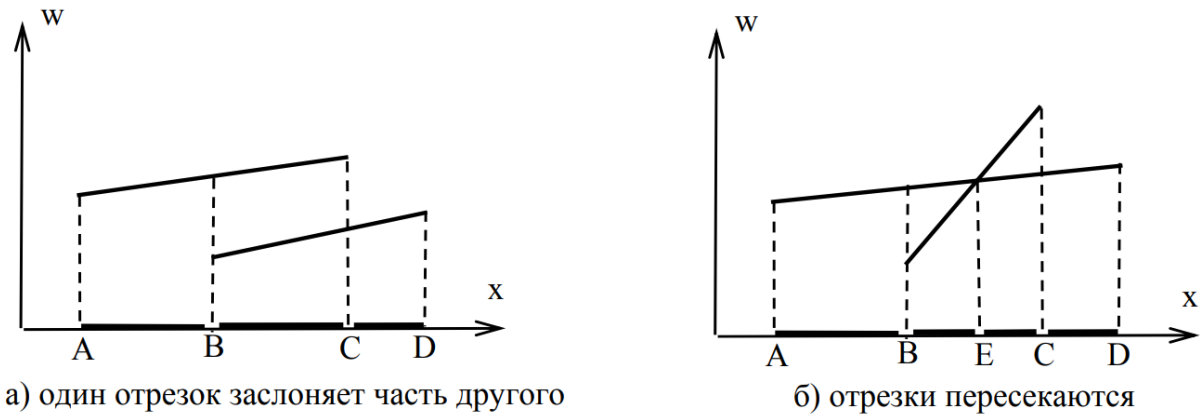

Рисунок 27 – Определение заслонённых частей отрезков.

Чтобы определить, какие части видимы на втором интервале, нужно рассчитать глубину точек отрезков на его концах. Для двух точек она известна, так как точки являются концами отрезков, для двух оставшихся можно рассчитать, используя уравнение прямой:

$$w = w_1 + (w_2 - w_1) \cdot \frac{x - x_1}{x_2 - x_1},$$

где x_1, w_1, x_2, w_2 – координаты двух концов отрезка, x, w – координаты точки пересечения. Если глубина обеих точек одного отрезка больше, чем у точек второго, то второй отрезок заслонён первым на этом интервале (см. рис. 27а). Если в начале интервала больше глубина одного отрезка, а в конце – другого, то отрезки пересекаются (см. рис. 27б). В этом случае необходимо найти точку пересечения отрезков, например, приравняв вышеприведённое уравнение прямой для прямых обоих отрезков. Интервал ***B-C*** поделится точкой пересечения ***E*** на два интервала (***B-E***, ***E-C***). На первом будет виден отрезок, глубина которого в точке ***B*** больше, чем у другого, на втором – отрезок, глубина которого в точке ***C*** больше, чем у другого. [3]

1.3.10 Выводы

Таблица 1 представляет собой сравнительный анализ алгоритмов удаления невидимых линий по основным критериям, важным для задачи создания и редактирования 3D-сцен. Алгоритмы представлены в столбцах, а критерии их оценки — в строках.

Обозначения алгоритмов:

- ПГ — Алгоритм плавающего горизонта.
- Роб — Алгоритм Робертса.
- Вар — Алгоритм Варнака.
- В-А — Алгоритм Вейлера-Азертона.
- Худ — Алгоритм художника.
- ТЛ — Алгоритм трассировки лучей.
- Z-Б — Алгоритм Z-буфера.
- ПС — Методы построчного сканирования.

Обозначения уровней оценки:

- В — Высокий уровень.
- С — Средний уровень.
- Н — Низкий уровень.
- ОВ — Очень высокий уровень.
- Огр — Ограниченная функциональность.

Таблица позволяет легко сравнить алгоритмы по таким критериям, как простота реализации, скорость работы, точность результата, объём используемой памяти и уровень интерактивности. Данные сокращения обеспечивают компактное представление информации, сохраняя при этом её содержательность.

Критерий	ПГ	Роб	Вар	В-А	Худ	ТЛ	Z-Б	ПС
Простота	С	Н	С	Н	В	Н	В	С
Скорость	В	С	С	Н	В	Н	Н	В
Точность	С	В	В	ОВ	С	ОВ	В	С
Память	Н	Н	Н	С	Н	В	Н	Н
Интерактивность	Н	Н	Огр	Огр	В	Огр	С	В

Таблица 1 – Сравнение алгоритмов удаления невидимых линий по критериям

Алгоритм Z-буфера выбран для задачи построения цифровой модели помещения по следующим причинам:

1. Точность: Z-буфер позволяет точно определять видимость объектов, так как каждый пиксель имеет соответствующую глубину, что важно для сцены, состоящей из параллелепипедов.

2. Скорость: Несмотря на то, что Z-буфер может требовать значительных вычислительных ресурсов для сложных сцен, он остаётся достаточно быстрым для большинства случаев. Однако для сложных сцен с высоким разрешением может потребоваться больше времени на перерасчёт и больше памяти из-за хранения матрицы глубины для каждого пикселя.

3. Память: Хотя Z-буфер требует много памяти для хранения данных о глубине, это компенсируется его простотой и высокой точностью. Однако в интерактивных приложениях, где сцена изменяется динамически, перерасчёт для каждого пикселя может быть медленным.

Таким образом, Z-буфер подходит для решения задачи, но требует оптимизаций, если сцена будет сложной или с большим количеством объектов.

1.4. Алгоритмы отрисовки теней

1.4.1 Введение

Тени являются неотъемлемой частью реалистичных изображений в компьютерной графике. Они помогают передать пространственные соотношения между объектами сцены, добавляют глубину и реализм визуализации. Существует множество методов отрисовки теней, которые варьируются по сложности и требуемым вычислительным ресурсам. В связи с выбором алгоритма Z-буфера для удаления невидимых линий и плоскостей, удобно использовать его и для алгоритма построения теней. Такой подход позволяет сохранить простоту структуры программы, исключить сложности при интеграции двух разных методов и, как следствие, сократить время, необходимое на отладку алгоритма.

1.4.2 Принципы работы Z-буфера

Алгоритм Z-буфера основывается на хранении информации о глубине объектов сцены для каждого пикселя экрана. При расчёте теней алгоритм проходит два этапа:

1. Первый проход: Сцена анализируется с позиции источника света. Вычисляются точки, видимые со стороны источника, и их глубины заносятся в теневой Z-буфер.

2. Второй проход: Сцена визуализируется с позиции наблюдателя. Для каждого пикселя проверяется, находится ли он в тени, путём сравнения его координат с данными теневого Z-буфера.

Преимущества Z-буфера для отрисовки теней:

- Универсальность: метод подходит для сцен любой сложности, включая динамические сцены с движущимися объектами и источниками света.
- Реализация в реальном времени: алгоритм хорошо масштабируется на современных графических процессорах, обеспечивая приемлемую скорость работы.
- Простота интеграции: Z-буфер легко адаптируется для построения теней, поскольку он изначально используется для удаления невидимых поверхностей.

Ограничения метода:

- Артефакты на границах теней: из-за дискретизации возможны погрешности, такие как акне (z-fighting) или ступенчатые края.
- Сложность мягких теней: алгоритм изначально рассчитан на жёсткие тени. Реализация эффектов мягкости требует дополнительных вычислений.

1.4.3 Выводы

Алгоритм Z-буфера является эффективным выбором для построения теней в условиях, когда требуется высокая скорость и приемлемое качество визуализации. Несмотря на некоторые ограничения, он остаётся стандартным методом, широко применяемым в компьютерной графике благодаря простоте и универсальности. [10]

1.5. Алгоритмы затенения

1.5.1 Введение

В компьютерной графике широко используются алгоритмы затенения для наложения освещения на объекты сцены. Рассмотрим две наиболее популярные модели: затенения по Гуро (Gouraud shading) и затенения по Фонгу (Phong shading).

1.5.2 Затенение по Гуро

Модель затенения по Гуро предполагает линейную интерполяцию цвета между вершинами полигона. Это требует, чтобы нормали были заданы в вершинах, так как цвет рассчитывается на основе нормалей. Преимущества и недостатки модели:

- Преимущества:
  - Высокая скорость работы: дорогостоящие расчёты освещения производятся только в вершинах.
  - Легко оптимизируется благодаря линейной интерполяции.
- Недостатки:
  - Потеря бликов на поверхности при низкой детализации модели.
  - Размытые отражения, особенно заметные на крупных полигонах.

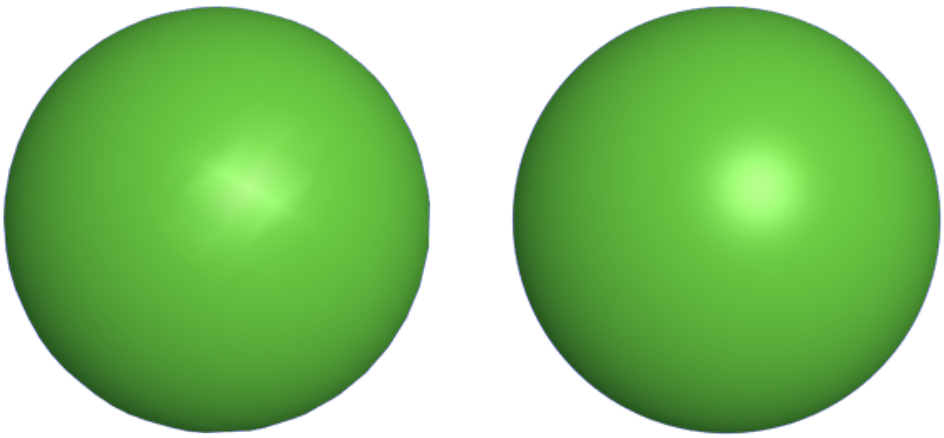

Рисунок 28 – Сфера с затенением по Гуро: около 2000 (слева) и 32000 треугольников (справа).

1.5.3 Затенение по Фонгу

Затенение по Фонгу работает иначе: между вершинами интерполируется не цвет, а нормаль, а освещение рассчитывается для каждого пикселя. Это позволяет добиться более высокой визуальной точности:

- Преимущества:
  - Качественное изображение без смазывания бликов.
  - Высокая степень реализма благодаря попиксельному расчёту освещения.
- Недостатки:
  - Высокие вычислительные затраты, особенно для сцен с высоким разрешением.

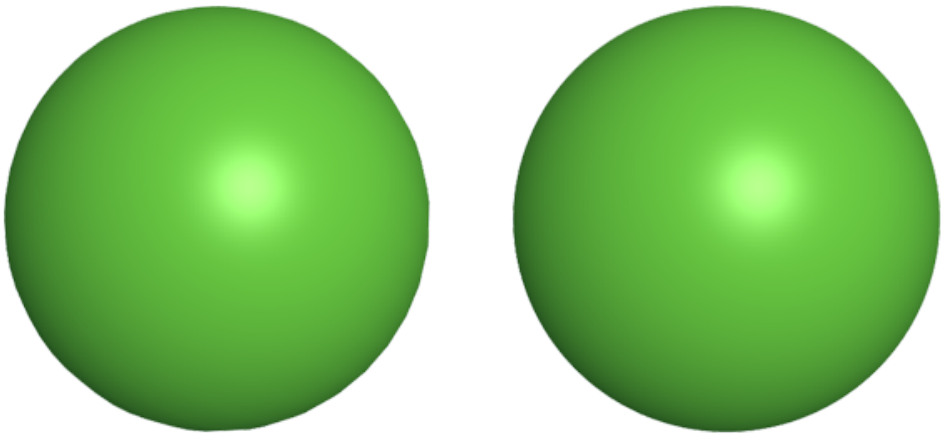

Рисунок 29 – Сфера с затенением по Фонгу: около 2000 (слева) и 32000 треугольников (справа).

1.5.4 Выводы

Модель затенения по Гуро обладает достаточной скоростью для многих задач, однако затенение по Фонгу обеспечивает значительно более реалистичное изображение, особенно для сложных сцен. Благодаря современным графическим процессорам затенение по Фонгу стало стандартом в игровой индустрии. [9]

2. Конструкторская часть

2.1. Требования к программному обеспечению

На основании введения и поставленных задач сформулированы следующие требования к разрабатываемому программному обеспечению для построения 3D сцен помещений:

2.1.1 Функциональные требования

1. Моделирование помещений и объектов:
    - Возможность добавления базовых 3D объектов: стен, окон, дверей.
    - Поддержка настройки параметров объектов (длина, ширина, высота, аналогично для отверстия).
    - Функции перемещения, изменения параметров, вращения и удаления объектов на сцене.
    - Обеспечение точного размещения объектов с проверкой пересечений и соблюдением заданных границ.
2. Управление камерой:
    - Предоставление функций управления камерой для осмотра сцены:
        - Вращение вокруг всех трёх осей.
        - Изменение масштаба для просмотра деталей и общей картины.
3. Работа со сценами:
    - Сохранение текущей сцены в файл.
    - Загрузка ранее сохранённых сцен для продолжения работы.
4. Пользовательский интерфейс:
    - Разработка интуитивно понятного интерфейса для работы с объектами и сценой.
    - Наличие панелей инструментов для выбора объектов, редактирования их параметров и управления камерой.
    - Поддержка визуализации сцены в реальном времени.
5. Производительность:
    - Плавная работа с увеличением количества объектов на сцене.
    - Оптимизация обработки сцен для предотвращения задержек и рывков при взаимодействии.

6. Тестирование:

    - Проверка корректности работы с объектами, сценой и камерой в различных условиях.
    - Тестирование функций сохранения и загрузки сцен.

2.1.2 Нефункциональные требования

1. Платформенная независимость:

    - Программа должна поддерживать работу на основных операционных системах (Windows, macOS, Linux).

2. Простота и удобство использования:

    - Дружелюбный интерфейс, не требующий глубоких технических знаний.

3. Надёжность:

    - Минимизация сбоев программы при работе с большими сценами.
    - Обработка ошибок и предупреждений при неправильных действиях пользователя.

4. Расширяемость:

    - Возможность добавления новых типов объектов и функций без значительных изменений в структуре программы.

5. Безопасность данных:

    - Сохранение и загрузка файлов без потери данных и повреждения структуры сцены.

2.1.3 Ожидаемые результаты

Программное обеспечение должно позволить быстро и эффективно моделировать 3D сцены помещений, обеспечивая интерактивное добавление и редактирование объектов. Оно станет полезным инструментом для анализа, планирования и обучения в области безопасности и оперативного реагирования.

2.2. Общий алгоритм решения поставленной задачи

1. Определение параметров сцены:

    - Установить размеры сцены, на которой будут размещаться объекты.

2. Размещение элементов сцены:

- Разместить необходимые объекты (стены, окна, двери и другие элементы) и определить источники света.

3. Визуализация сцены с тенями:

   - На основе текущего положения наблюдателя использовать модифицированный алгоритм, основанный на Z-буфере, для расчёта теней, падающих от объектов сцены.
   - Выполнить рендеринг сцены с учётом теней и освещения.

2.3. Алгоритм Z-буфера удаления невидимых линий

Алгоритм Z-буфера удаления невидимых линий на псевдокоде:

1. Заполнить буфер кадра фоновым значением интенсивности или цвета.
2. Заполнить Z-буфер минимальным значением глубины z.
3. Преобразовать каждый многоугольник в растровую форму в произвольном порядке.
4. Для каждого пикселя (x,y) в многоугольнике вычислить его глубину $z(x,y)$.
5. Сравнить вычисленную глубину $z(x,y)$ со значением в Z-буфере в этой же позиции $Z_{\text{буфер}}(x,y)$.
6. Запись в буфер кадра:

   - Если $z(x,y) > Z_{\text{буфер}}(x,y)$, то записать атрибуты многоугольника (интенсивность, цвет и т. п.) в буфер кадра и заменить значение в Z-буфере на $z(x,y)$.
   - В противном случае никаких действий не производить. [?]

Блок-схема алгоритма Z-буфера удаления невидимых линий представлена на рисунке 30.

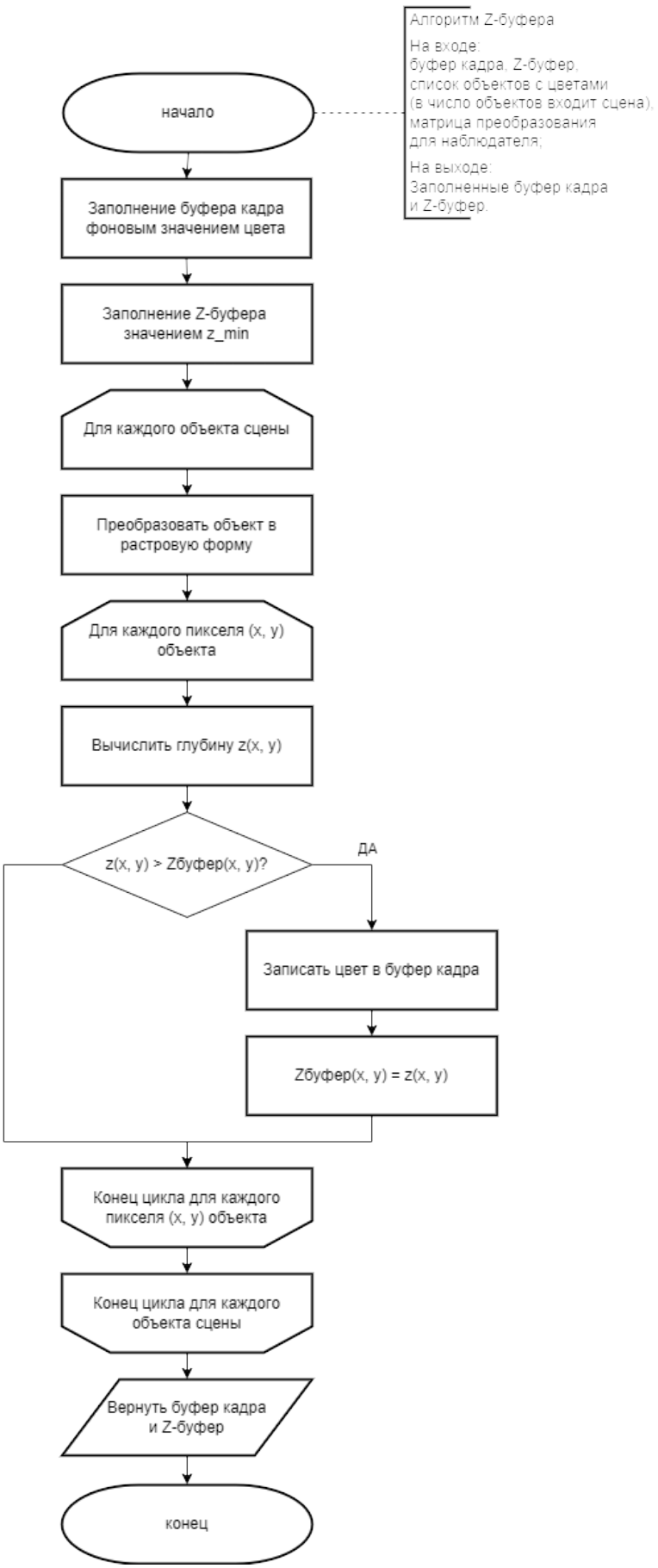

Рисунок 30 – Блок-схема алгоритма Z-буфера удаления невидимых линий.

2.4. Алгоритм Z-буфера с отрисовкой теней

Алгоритм Z-буфера с отрисовкой теней на псевдокоде:

1. Выполнить расчет Z-буфера для основной сцены:
   (a) Заполнить буфер кадра фоновым значением интенсивности или цвета.
   (b) Заполнить Z-буфер минимальным значением глубины z.
   (c) Преобразовать каждый многоугольник в растровую форму в произвольном порядке.
   (d) Для каждого пикселя (x,y) в многоугольнике:
   - Вычислить глубину пикселя $z(x,y)$.
   - Сравнить $z(x,y)$ с текущим значением $Z_{\text{буфер}}(x,y)$.
   - Если $z(x,y) > Z_{\text{буфер}}(x,y)$, записать атрибуты пикселя в буфер кадра и обновить $Z_{\text{буфер}}(x,y)$.
2. Выполнить расчет Z-буфера для вида из источника света:
   (a) Заполнить буфер кадра фоновым значением.
   (b) Заполнить Z-буфер минимальным значением глубины z.
   (c) Преобразовать каждый многоугольник в растровую форму относительно вида из источника света.
   (d) Для каждого пикселя вычислить его глубину $z(x,y)$ и записать в Z-буфер для создания карты теней.
3. Наложить матрицу теней на основную сцену:
   (a) Для каждого пикселя (x,y) в Z-буфере основной сцены:
   - Извлечь глубину $z(x,y)$ и преобразовать координаты (x,y,z) в координаты вида из источника света (x',y',z').
   - Если x',y' находятся в пределах карты теней, сравнить $z(x,y)$ с глубиной из карты теней $z_{\text{light}}(x',y')$.
   - Если $z(x,y) > z_{\text{light}}(x',y') + \text{EPS}$, уменьшить яркость пикселя или применить цвет тени.

Блок-схема алгоритма Z-буфера с отрисовкой теней представлена на рисунке 31.

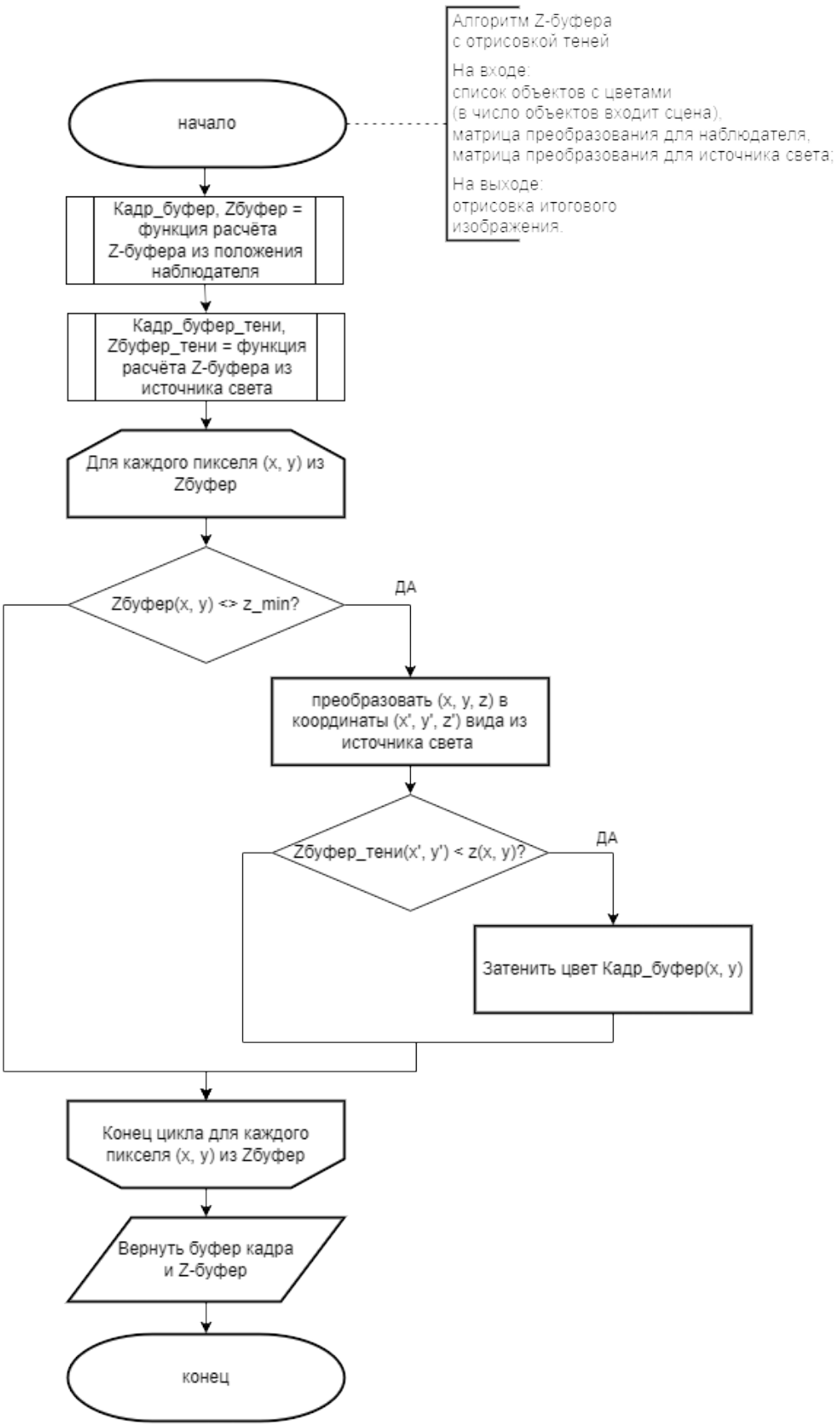

Рисунок 31 – Блок-схема алгоритма Z-буфера с отрисовкой теней.

2.5. Представление объектов в программном обеспечении

1. Точка трёхмерного пространства

Представлена координатами по осям x, y, z.

2. Стена
   Задаётся как параллелепипед, параметры которого включают длину, высоту, ширину и координаты начальной точки.

3. Окно
   Подвид стены, представлено параллелепипедом с параметрами длины, высоты, ширины и расположения (координаты начальной точки).

4. Дверь
   Аналогично окну, задаётся параллелепипедом с уникальными параметрами длины, высоты, ширины и позиции в пространстве.

5. Сцена
   Представлена набором плиток (квадратных полигонов), каждая из которых определяется координатами (x, y, z), где $z = 0$ для пола.

6. Камера
   Описывается матрицей преобразования, включающей параметры положения камеры (координаты) и направления взгляда (углы поворота по осям x, y, z).

7. Источник света
   Представлен аналогично камере, с использованием матрицы преобразования, включающей параметры положения источника света (углы поворота по осям x, y, z).

2.6. Выводы

В рамках конструкторской части были сформулированы основные требования к программному обеспечению для моделирования 3D сцен помещений. Также были рассмотрены два ключевых алгоритма рендеринга: алгоритм Z-буфера для удаления невидимых линий и его расширение с отрисовкой теней. Оба алгоритма направлены на создание более реалистичной визуализации сцен с учетом взаимодействия объектов и источника света. Представление объектов в программном обеспечении включает точное описание 3D объектов (стены, окна, двери) и их параметров, а также обработку сцены, камеры и источника света с использованием матрицы преобразования.

Таким образом, работа направлена на создание системы, которая будет эффективно решать задачи моделирования помещений с высокой степенью взаимодействия и визуализации, обеспечивая удобство работы и надежность при взаимодействии с пользователем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Роджерс Д. Алгоритмические основы машинной графики. — М.: Мир, 1989. — 512 с.

2. ИТЕР. Модель системы защиты [Электронный ресурс]. — Режим доступа: https://iter.ru/model_sfz.html#section_9, дата обращения: 21.09.2024.

3. Польский С. В. Компьютерная графика: учеб.-метод. пособие. — М.: ГОУ ВПО МГУЛ, 2008. — 38 с.

4. Шикин Е. В., Боресков А. В. Компьютерная графика. Полигональные модели. — М.: Диалог-МИФИ, 2001. — 464 с.

5. Турлапов В. Е. Удаление невидимых поверхностей. Оптимизация. Тени [Электронный ресурс]. — Нижегородский государственный университет имени Н. И. Лобачевского, 2013. — Режим доступа: http://www.graph.unn.ru/rus/materials/CG/CG13_HSROptimization.pdf, дата обращения: 21.11.2024.

6. Алгоритм, использующий z–буфер [Электронный ресурс]. — Режим доступа: https://compgraph.tpu.ru/zbuffer.htm, дата обращения: 21.11.2024.

7. Шикин Е. В., Боресков А. В. Компьютерная графика. Динамика, реалистические изображения. — М.: Диалог-МИФИ, 1995. — 288 с.

8. Дерягина О. В., Корниенко В. В., Лагерь А. И., Скоробогатова Т. Е. Компьютерная графика: учебное пособие. — Красноярск: Красноярский государственный аграрный университет, 2014. — 322 с. — Режим доступа: http://www.kgau.ru/distance/etf_06/komp-grafika/index.htm, дата обращения: 21.11.2024.

9. Компьютерная графика: теория, алгоритмы, примеры на C++ и OpenGL [Электронный ресурс] / Освещение и модель затенения. URL: https://compgraphics.info/3D/lighting/shading_model.php (дата обращения: 21.11.2024).

10. Романюк А.Н., Куринный М.В. Алгоритмы построения теней // Компьютеры + программы. – 2000. – № 8–9. – С. 5–6.