

# СОДЕРЖАНИЕ

<b>ОПРЕДЕЛЕНИЯ</b>	<b>6</b>
<b>ВВЕДЕНИЕ</b>	<b>7</b>
<b>1 Аналитическая часть</b>	<b>8</b>
1.1 Формализация объектов сцены	8
1.2 Анализ способов представления модели	8
1.3 Анализ способов представления поверхностной модели, заданной полигональной сеткой	11
1.4 Анализ алгоритмов удаления невидимых линий и поверхностей	14
1.4.1 Алгоритм Робертса	14
1.4.2 Алгоритм Варнока	16
1.4.3 Алгоритм трассировки лучей	18
1.4.4 Алгоритм обратной трассировки лучей	19
1.4.5 Алгоритм художника	19
1.4.6 Алгоритм Z-буфера	21
1.4.7 Выбор оптимального алгоритма	22
1.5 Анализ модели освещения	22
1.5.1 Модель Ламберта	22
1.5.2 Модель Фонга	23
1.5.3 Модель Блинна-Фонга	25
1.5.4 Выбор оптимальной модели освещения	26
1.6 Анализ алгоритмов закрашивания	26
1.6.1 Простая закрашка	26
1.6.2 Закрашка Гуро	27
1.6.3 Закрашка Фонга	27

1.6.4	Выбор оптимального алгоритма закрашки . . . . .	28
1.7	Анализ алгоритма построения теней . . . . .	28
1.8	Вывод . . . . .	29
<b>2</b>	<b>Конструкторская часть . . . . .</b>	<b>30</b>
2.1	Требования к программе . . . . .	30
2.2	Общий алгоритм визуализации сцены . . . . .	30
2.3	Аффинные преобразования . . . . .	34
2.4	Камера . . . . .	35
2.4.1	Пирамида видимого пространства . . . . .	35
2.4.2	Пространство модели . . . . .	36
2.4.3	Мировое пространство . . . . .	36
2.4.4	Пространство камеры . . . . .	37
2.4.5	Пространство отсечения . . . . .	39
2.4.6	Пространство экрана . . . . .	40
2.4.7	Алгоритм получения изображения от лица камеры . . . . .	40
2.5	Невидимые грани . . . . .	42
2.6	Алгоритм Z-буфера . . . . .	42
2.7	Выбор типов и структур данных . . . . .	45
2.8	Диаграмма классов . . . . .	46
2.9	Вывод . . . . .	47
<b>3</b>	<b>Технологическая часть . . . . .</b>	<b>48</b>
3.1	Средства реализации . . . . .	48
3.2	Структура программы . . . . .	48
3.3	Исходный код . . . . .	54
3.4	Интерфейс программы . . . . .	54
3.5	Вывод . . . . .	59
<b>4</b>	<b>Исследовательская часть . . . . .</b>	<b>60</b>
4.1	Технические характеристики . . . . .	60
4.2	Проведение исследования . . . . .	60
4.2.1	Зависимость времени визуализации от количества поли- гонов . . . . .	60

4.3 Вывод . . . . .	61
<b>ЗАКЛЮЧЕНИЕ . . . . .</b>	<b>63</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ . . . . .</b>	<b>64</b>
<b>Приложение А . . . . .</b>	<b>65</b>

# ОПРЕДЕЛЕНИЯ

Рендеринг (англ. Rendering) – конечный процесс создания реального 2D-изображения или анимации из подготовленной сцены.

Грань – замкнутое множество рёбер.

Полигон – набор компланарных граней.

# ВВЕДЕНИЕ

Компьютерная графика является одной из самых динамично развивающихся областей информационных технологий, играющей ключевую роль в современном обществе. Она охватывает широкий спектр приложений, включая игры, анимацию, визуализацию данных, дизайн, архитектуру и медицину.

Целью данной курсовой работы является разработка программы композиции и визуализации трехмерных многогранных примитивов с учётом их геометрических и оптических параметров, вводимых пользователем. Для удобства восприятия должны присутствовать эффект глубины визуализируемой сцены, опции перемещения камеры, настройка положения источника света и возможность изменения его спектральных характеристик.

Чтобы достичь данной цели, необходимо выполнить следующие задачи:

- описать объекты сцены;
- проанализировать существующие алгоритмы компьютерной графики для генерации реалистичных моделей и трехмерной сцены;
- выбрать наиболее подходящие алгоритмы для достижения поставленной цели;
- спроектировать архитектуру и графический интерфейс приложения;
- выбрать средства реализации программного обеспечения;
- реализовать выбранные алгоритмы и структуры данных;
- провести исследование быстродействия разработанного приложения.

# 1 Аналитическая часть

В этой части рассматриваются объекты сцены, способы их представления и существующие алгоритмы создания изображений, а также осуществляется выбор наиболее подходящих из этих алгоритмов для дальнейшего применения.

## 1.1 Формализация объектов сцены

Сцена является набором объектов, включающим в себя:

- 1) объекты сцены – геометрические многогранные примитивы, расположенные в пространстве сцены, такие как, куб, прямая призма треугольная пирамида. Каждый примитив состоит из граней, сформированных из точек соединенных ребрами. Геометрические характеристики примитивов задаются параметрами такими, как длина ребра основания, высота, радиусы описанных окружностей нижнего и верхнего основания, количество боковых граней. Спектральные характеристики так же задаются параметрами такими, как коэффициенты отражения и блеска, цвет.
- 2) источник света – пространственный вектор, указывающий направление света. Свет всегда направлен вдоль положительной оси  $Z$ , его положение в пространстве сцены задается пользователем.
- 3) камера – пространственный вектор, указывающий направление просмотра.

## 1.2 Анализ способов представления модели

- 1) *каркасная модель* – простейший способ представления моделей, основанный на использовании точек и рёбер, содержащий минимум информации и имеющий ряд ограничений. Основным недостатком является неоднозначность, так как невозможно четко определить ориентацию и видимость, что может привести к непредсказуемым результатам. Удаление скрытых линий требует ручного редактирования, что может разрушить целостность модели. Также каркасные модели не поддерживают технику тонового изображения, так как затенение применяется к граням, а не к ребрам, что ограничивает их использование в задачах, требующих визуальной глубины и реалистичности. Несмотря на эти ограничения, каркас-

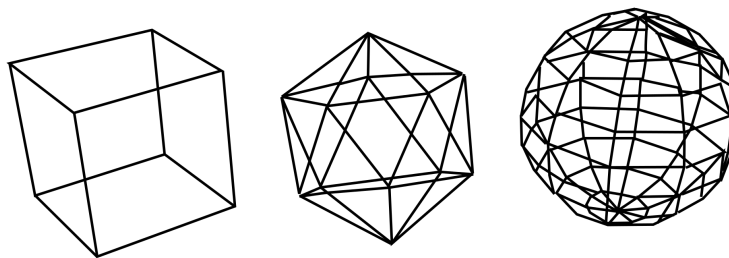


Рисунок 1.1 — Каркасные модели

ные модели требуют меньше памяти и легки в отрисовке. Пример каркасных моделей приведён на рисунке 1.1.

2) *поверхностная модель* – способ представления моделей, основанный на использовании точек, линий и поверхностей, что делает его более гибким и многофункциональным по сравнению с каркасной моделью. Он позволяет изображать сложные криволинейные грани, создавать тоновые трехмерные изображения и выявлять особенности, такие как отверстия. Этот метод обеспечивает высокое качество изображений. Поверхностную модель можно задать двумя способами:

- *параметрическое представление* – вид поверхностной модели, требующий вычисления функции, зависящей от параметра, но его использование затруднено в сценах без поверхностей вращения;
- *полигональная сетка* – вид поверхностной модели, представленный совокупностью вершин, рёбер и граней. Гранями обычно являются треугольники, четырёхугольники или другие простые выпуклые многоугольники, но могут также являться многоугольниками с отверстиями. Пример поверхностной модели, заданной полигональной сеткой, приведён на рисунке 1.2.

Однако у поверхностной модели есть недостаток: она не предоставляет информации о том, с какой стороны поверхности находится материал.

3) *твердотельная модель* – способ представления моделей, включающий информацию о внутренней структуре и расположении материала, в отличие от поверхностной модели, что достигается указанием направления внутренней нормали. Твердотельные модели обладают неоспоримыми преимуществами, включая полное определение объемной формы с разграничением внешних и внутренних областей, что позволяет избежать

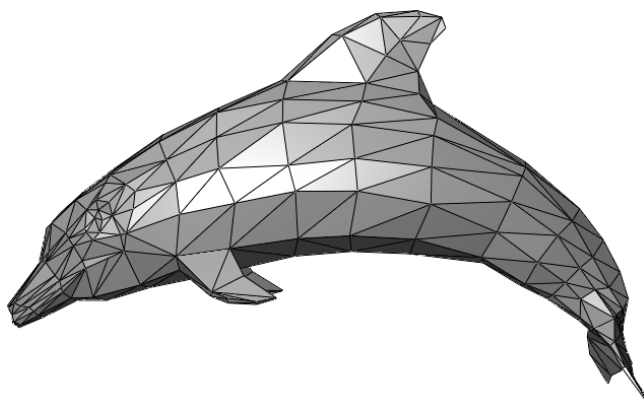


Рисунок 1.2 — Поверхностная модель, заданная полигональной сеткой

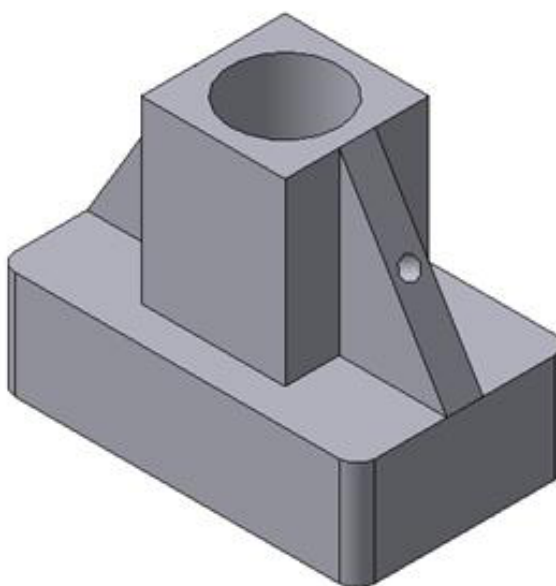


Рисунок 1.3 — Твёрдотельная модель

нежелательных взаимовлияний компонентов. Они обеспечивают автоматическое удаление скрытых линий и построение трехмерных разрезов, что важно для анализа сложных композиций. Пример твердотельной модели приведён на рисунке 1.3.

Для выполнения данной работы каркасная модель не подходит, так как она может исказить восприятие форм объекта. Твёрдотельная модель так же не является оптимальной, поскольку в этой работе не требуется информация о материале и его расположении. Таким образом, остается только поверхностная модель как наилучший выбор.

Поверхностная модель будет задана полигональной сеткой, так как в этой работе не будет обработки поверхностей вращения и использование парамет-



рического представления избыточно.

### 1.3 Анализ способов представления поверхностной модели, заданной полигональной сеткой

Объекты, созданные с помощью полигональных сеток, должны хранить разные типы элементов, такие как вершины, рёбра, грани, полигоны и поверхности. Во многих случаях хранятся лишь вершины, рёбра и либо грани, либо полигоны (рис. 1.4). Грани могут быть как трехсторонними, так и четырехсторонними.

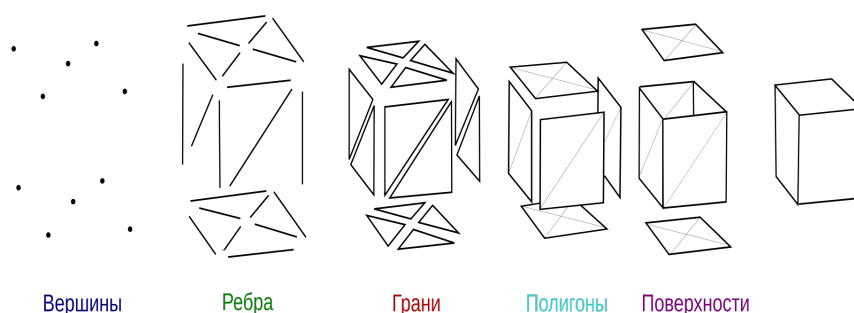


Рисунок 1.4 — Элементы моделирования сетки

- 1) *вершинное представление* – простейший способ представления, описывающий объект как набор вершин, соединённых с другими вершинами. Хотя это и простейший способ представления, он не так широко используется, поскольку не предоставляет явной информации о гранях и рёбрах. Это означает, что для генерации списка граней, необходимого для рендеринга, нужно обойти все данные, что усложняет выполнение операций с рёбрами и гранями. Пример полигональной сетки, заданной вершинным представлением приведён на рисунке 1.5.
- 2) *список граней* – способ представления, описывающий объект как множество вершин и граней, который является наиболее распространённым представлением для современного графического оборудования. Это представление упрощает моделирование, позволяя легко находить грани, окружающие конкретные вершины. Пример полигональной сетки, заданной списком граней приведён на рисунке 1.6.
- 3) *«крылатое» представление* – способ представления, описывающий объект как множество вершин, граней и рёбер, что обеспечивает высокую

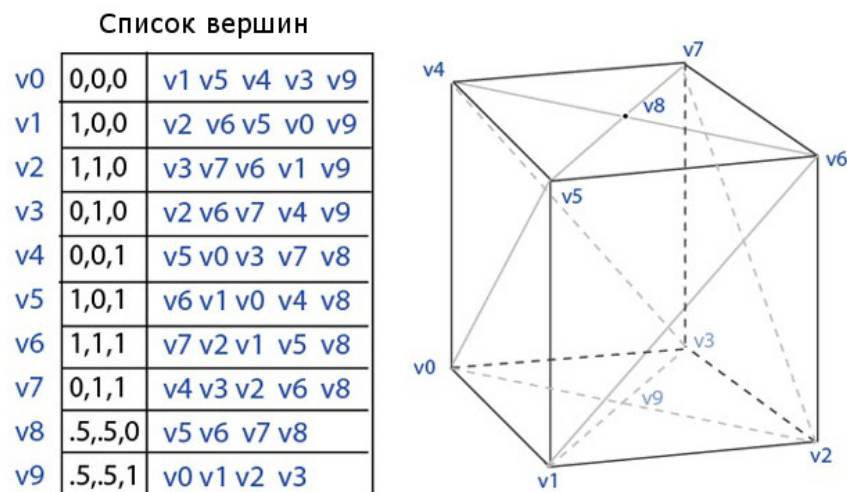


Рисунок 1.5 — Вершинное представление

гибкость в динамическом изменении геометрии. «Крылатое» представление эффективно решает задачу обхода от ребра к ребру, предоставляя упорядоченное множество граней вокруг каждого ребра. Оно включает информацию о двух конечных вершинах, двух гранях и четырёх ближайших рёбрах. Пример полигональной сетки, заданной «крылатым» представлением приведён на рисунке 1.7.

Для хранения полигональной сетки будет использован метод, основанный на списке граней, что обеспечивает ясное представление о гранях. Этот подход способствует эффективному преобразованию моделей, так как структура включает в себя список вершин.

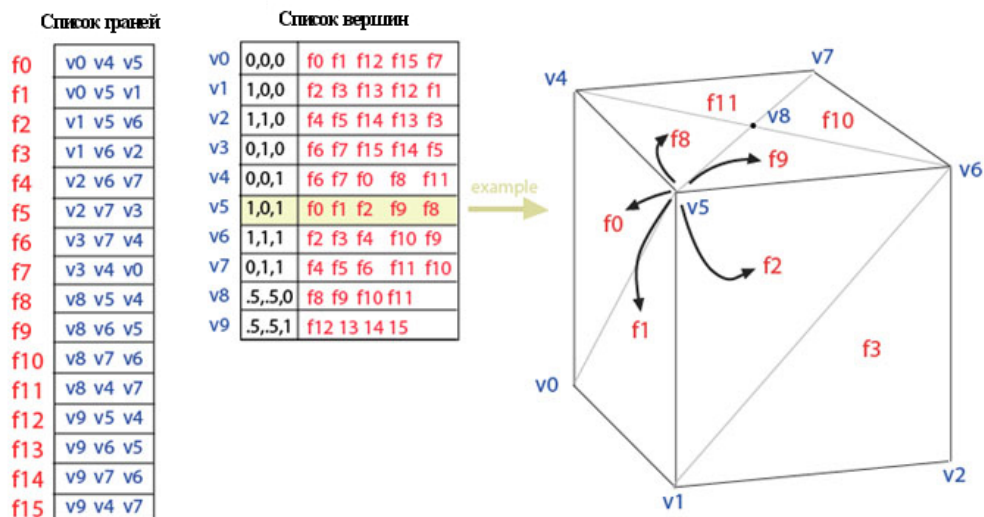


Рисунок 1.6 — Список граней

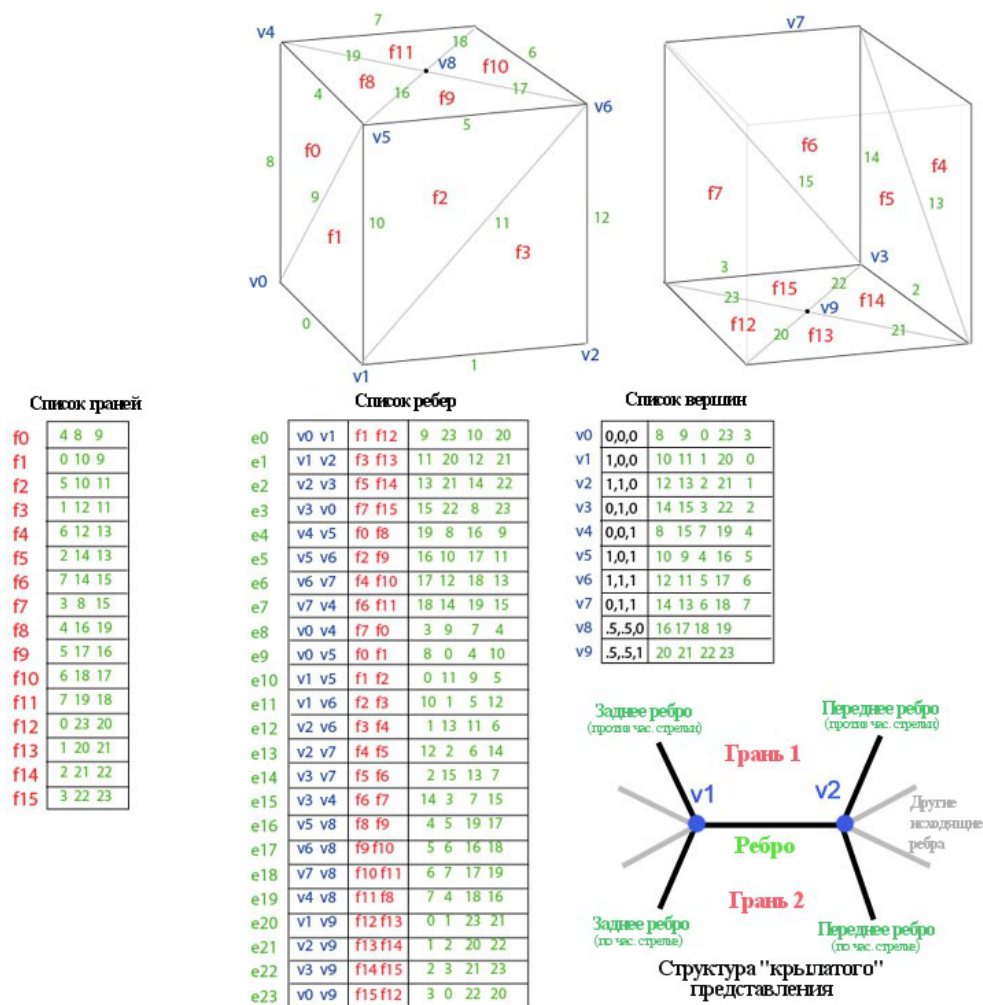


Рисунок 1.7 — «Крылатое» представление

## 1.4 Анализ алгоритмов удаления невидимых линий и поверхностей

Главной целью при создании реалистичного изображения является удаление объектов или их частей, которые не видны наблюдателю из-за перекрытия другими объектами. Для решения этой задачи выделяют две категории алгоритмов:

- *Алгоритмы, работающие в объектном пространстве*, привязаны к мировой или физической системе координат. Их результаты зависят только от точности вычислений и требуют значительных вычислительных ресурсов, что зависит от необходимой точности и сложности входной сцены. К этой категории относятся алгоритмы Робертса, алгоритмы со списком приоритетов и другие.
- *Алгоритмы, работающие в пространстве изображения*, ориентированы на систему координат экрана или картинной плоскости, на которую проецируются объекты. Объем вычислений в этой группе значительно меньше по сравнению с первой, и он зависит от разрешающей способности экрана и количества объектов в сцене. К основным алгоритмам этой группы относятся алгоритм Варнока, алгоритм Z-буфера и алгоритм трассировки лучей.

### 1.4.1 Алгоритм Робертса

Алгоритм Робертса — это первое известное решение задачи об удалении невидимых линий в трехмерной графике. Он представляет собой математически элегантный метод, работающий в объектном пространстве. Основная задача алгоритма заключается в удалении ребер и граней, которые экранируются самим объектом, а затем в сравнении оставшихся видимых ребер с другими объектами для определения их видимости.

Работа Алгоритма Робертса проходит в два этапа:

- определение нелицевых граней для каждого тела отдельно;
- определение и удаление невидимых ребер.

Для корректной работы данного алгоритма, необходимо выполнение следующих предварительных условий:

- тело ограничено плоскостями;
- нормали всех граней направлены внутрь тела;
- тело выпукло (невыпуклые тела должны быть разбиты на выпуклые части).

В этом алгоритме выпуклое многогранное тело с плоскими гранями должно представиться набором пересекающихся плоскостей. Уравнение произвольной плоскости в трехмерном пространстве имеет вид:

$$ax + by + cz + d = 0 \quad (1.1)$$

В матричной форме уравнение 1.1 выглядит так:

$$\begin{pmatrix} x & y & z & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = 0 \quad (1.2)$$

Тогда набор пересекающихся плоскостей состоит из матрицы коэффициентов уравнений плоскостей и называется *матрицей тела*:

$$V = \begin{pmatrix} a_1 & a_2 & \cdots & a_n \\ b_1 & b_2 & \cdots & b_n \\ c_1 & c_2 & \cdots & c_n \\ d_1 & d_2 & \cdots & d_n \end{pmatrix} \quad (1.3)$$

Корректное формирование матрицы тела предполагает, что любая точка внутри него должна находиться по положительную сторону от каждой грани. Если это условие не соблюдено для определенной грани, следует умножить соответствующий столбец матрицы на -1. Для проверки правильности можно взять точку, находящуюся внутри тела, координаты которой вычисляются как среднее значение координат всех его вершин.

Далее удаляются грани, которые скрыты телом. Для этого используется вектор взгляда  $E = (0 \ 0 \ -1 \ 0)$ , и чтобы определить невидимые грани, вектор умножается на матрицу тела  $V$ . Отрицательные компоненты результата указывают на невидимые грани.

Для выявления невидимых точек на ребре необходимо провести луч от наблюдателя к точке наблюдения. Если луч сталкивается с каким-либо объектом, то точка считается невидимой. Если объект действительно препятствует прохождению луча, он должен пересекать этот объект, оставаясь по положительную сторону от каждой его грани.

Вычислительная сложность алгоритма возрастает теоретически как квадрат числа объектов, то есть его главный недостаток – скорость работы. К тому же, этот алгоритм сложно реализуемый из-за его полностью математической структуры.

Основное преимущество данного алгоритма заключается в высокой точности вычислений, которая достигается благодаря работе в объектном пространстве, в отличие от большинства других алгоритмов.

### 1.4.2 Алгоритм Варнока

Алгоритм Варнока основывается на гипотезе о том, как человеческий глаз и мозг обрабатывают информацию в сцене. Эта гипотеза утверждает, что на обработку областей с низким информационным содержанием уходит значительно меньше времени и усилий, чем на области с высоким содержанием информации.

Конкретная реализация алгоритма зависит от метода разбиения окна и от критериев, используемых для определения простоты содержимого окна. В оригинальной версии алгоритма каждое окно делится на четыре равных подокна. Многоугольник в изображаемой сцене может быть классифицирован следующим образом (рис. 1.8):

- *внешний*, если он находится полностью вне окна;
- *внутренний*, если он находится полностью внутри окна;
- *пересекающий*, если он пересекает границу окна;
- *охватывающий*, если окно находится полностью внутри него.

Алгоритм можно описать в общих чертах следующим образом:

- 1) если все многоугольники сцены являются внешними по отношению к окну, то окно считается пустым, заполняется фоновым цветом и дальнейшему разбиению не подлежит.
- 2) если только один многоугольник пересекает окно и является внутренним, окно заполняется фоновым цветом, а сам многоугольник — своим

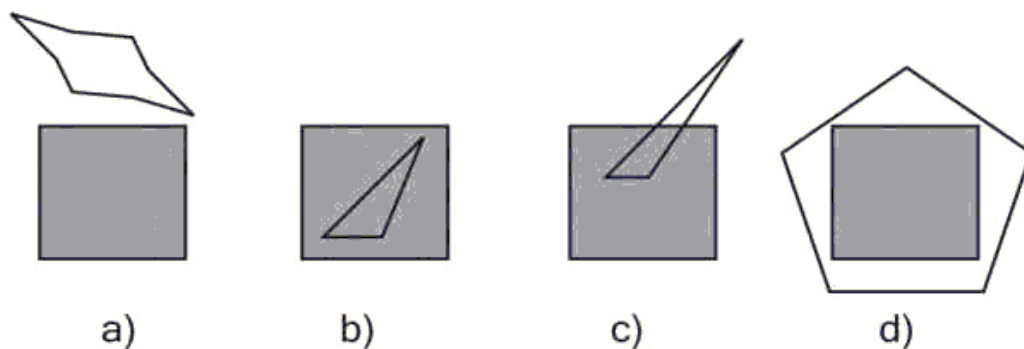


Рисунок 1.8 — Типы многоугольников в алгоритме Варнока: внешний (а), внутренний (b), пересекающий (с), охватывающий (d)

цветом.

- 3) если только один многоугольник пересекает окно и является пересекающим, окно заполняется фоновым цветом, а часть многоугольника, которая принадлежит окну, заполняется цветом этого многоугольника.
- 4) если только один многоугольник охватывает окно и нет других многоугольников, имеющих общие точки с окном, то окно заполняется цветом этого многоугольника.
- 5) если существует хотя бы один охватывающий окно многоугольник, из всех таких выбирается тот, который ближе всего к точке наблюдения, и окно заполняется его цветом.
- 6) в противном случае производится новое разбиение окна.

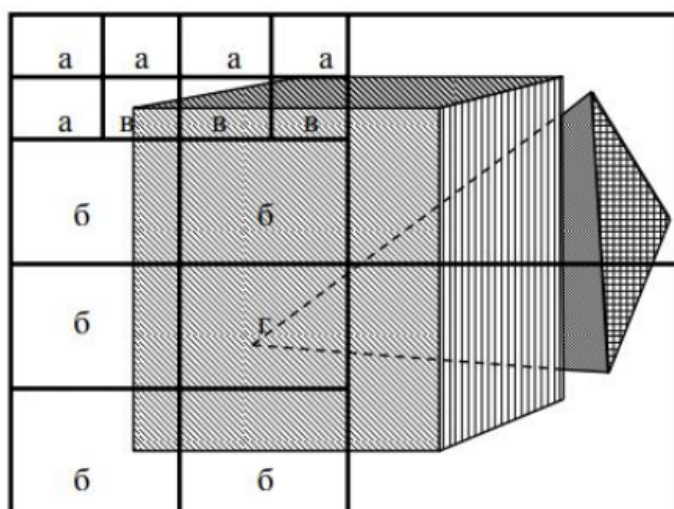


Рисунок 1.9 — Пример разбиения алгоритмом Варнока

Шаги 1–4 рассматривают случай пересечения окна только с одним много-

угольником, что помогает уменьшить количество подразбиений. Шаг 5 решает проблему удаления невидимых поверхностей, так как многоугольник, находящийся ближе к наблюдателю, экранирует остальные. Пример разбиения приведён на рисунке 1.9.

Основной недостаток алгоритма заключается в его рекурсивной работе. На каждом этапе он проверяет, какие грани видны, и если определение видимости нетривиально, окно разделяется на четыре секции, и анализ проводится отдельно для каждой из них.

### 1.4.3 Алгоритм трассировки лучей

Главная концепция алгоритма трассировки лучей состоит в том, что наблюдатель замечает объекты благодаря свету, исходящему от источников, который взаимодействует с объектами и, следуя законам оптики, достигает его глаза. В методе прямой трассировки создаются траектории лучей от всех источников света ко всем точкам объектов, что приводит к большому количеству обрабатываемых лучей и значительным вычислительным затратам, поскольку лишь небольшая часть лучей достигает точки наблюдения.

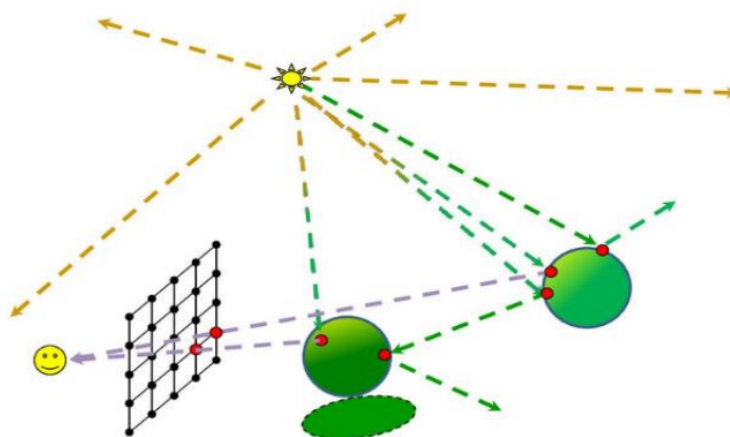


Рисунок 1.10 — Алгоритм трассировки лучей

Этот алгоритм подходит для создания статических сцен и моделирования различных оптических эффектов, таких как отражение и преломление.



### 1.4.4 Алгоритм обратной трассировки лучей

Идея алгоритма обратной трассировки лучей состоит в том, что наблюдатель видит объекты благодаря свету, исходящему от источников, который отражается от объектов и достигает его глаз. Однако отслеживание путей лучей от источника к наблюдателю неэффективно с вычислительной точки зрения, поэтому более целесообразно отслеживать их в обратном направлении — от наблюдателя к объектам. Лучи исходят из камеры, проходя через каждый пиксель сцены, после чего определяется их пересечение с объектами. Если пересечение найдено, рассчитывается интенсивность пикселя с учетом расположения источников света. Наблюдатель считается находящимся на положительной полуоси  $z$  в бесконечности, поэтому все лучи параллельны этой оси.

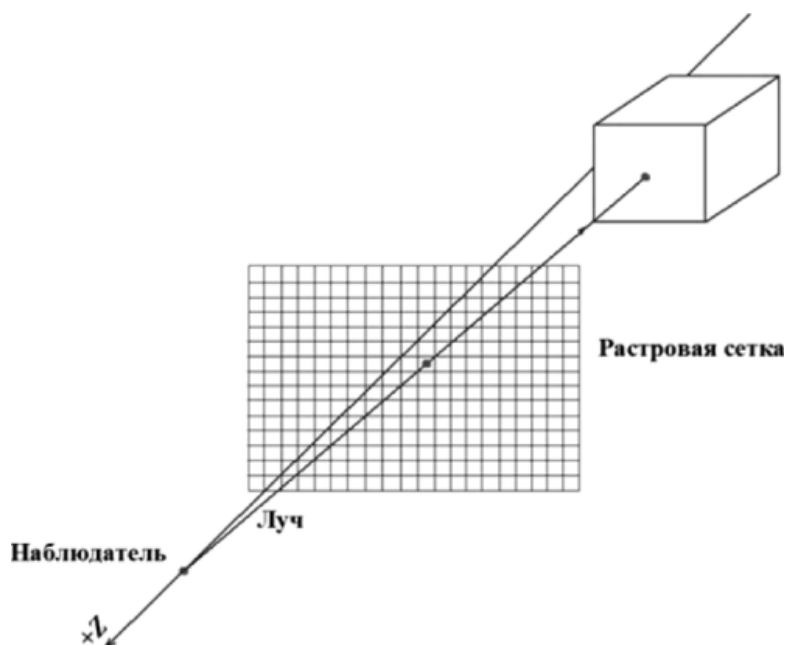


Рисунок 1.11 — Алгоритм обратной трассировки лучей

При этом, несмотря на большую эффективность алгоритма по сравнению с прямой трассировкой, он все же медленный из-за необходимости точно рассчитывать сложные аналитические выражения для нахождения пересечений с объектами.

### 1.4.5 Алгоритм художника

Алгоритм художника предполагает сортировку всех полигонов сцены по удаленности от наблюдателя. Объекты, находящиеся дальше от камеры, ренде-

рятся первыми, а затем последовательно добавляются более близкие объекты.

Например, в сцене с горами, лугом и деревьями, изображенными на рисунке 1.12, правильный порядок рендеринга будет: горы → луг → деревья.

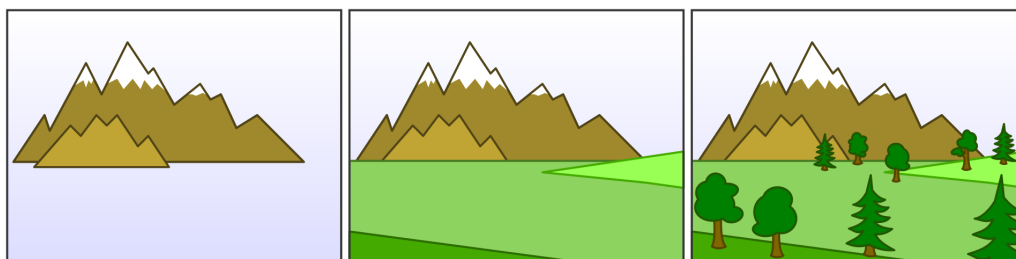


Рисунок 1.12 — Последовательная работа алгоритма художника

Несмотря на свою простоту, алгоритм художника имеет несколько значительных недостатков:

- если полигоны пересекаются, определить правильный порядок их рисования становится невозможно. Например, если три полигона накладываются друг на друга (рис. 1.13), то независимо от порядка сортировки, результат будет некорректным. Для решения этой проблемы может потребоваться разбивка конфликтующих полигонов на меньшие части, что усложняет реализацию;
- алгоритм также может прорисовывать области, которые впоследствии будут перекрыты другими объектами. Это приводит к ненужным затратам процессорного времени и ресурсов, так как рендерятся объекты, которые не будут видны в конечном результате.

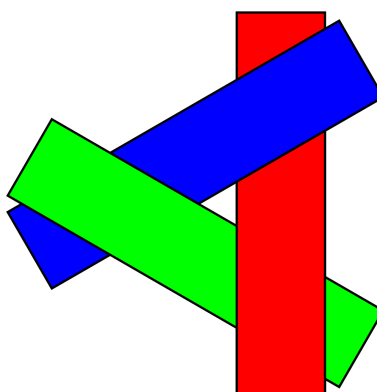


Рисунок 1.13 — Взаимноперекрывающиеся полигоны

Из-за этих недостатков был разработан алгоритм *Z*-буфера, который представляет собой более эффективный способ обработки видимости.

## 1.4.6 Алгоритм Z-буфера

Алгоритм  $z$ -буфера представляет собой один из наиболее простых способов удаления невидимых объектов, работающий в пространстве изображения. Он основан на концепции буфера кадра, который сохраняет интенсивность каждого пикселя, и дополнительно использует  $z$ -буфер для хранения координат глубины видимых пикселей. Во время работы алгоритм сравнивает глубину нового пикселя с уже сохраненной в  $z$ -буфере. Если новый пиксель находится ближе к наблюдателю, он добавляется в буфер кадра, а значение глубины обновляется. Если же он дальше, то ничего не происходит. Таким образом, алгоритм фактически ищет максимальное значение функции  $z(x, y)$ .

Формально алгоритм можно описать следующим образом:

- 1) инициализировать буфер кадра фоновым значением цвета или интенсивности.
- 2) заполнить  $z$ -буфер минимальными значениями глубины.
- 3) преобразовать многоугольники в растровую форму в произвольном порядке.
- 4) для каждого пикселя  $(x, y)$  в многоугольнике вычислить его глубину  $z(x, y)$ .
- 5) если  $z(x, y) > Z_{буфер}(x, y)$  (ближе к наблюдателю), обновить буфер кадра и  $z$ -буфер, иначе никаких действий не предпринимать.

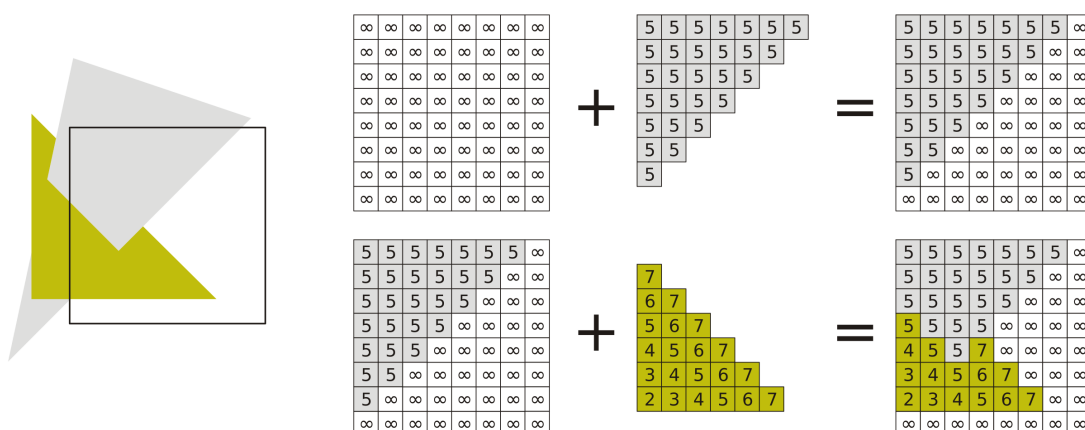


Рисунок 1.14 — Пример заполнения  $z$ -буфера

Главное преимущество этого метода – его простота. Он эффективно решает задачу удаления невидимых объектов и позволяет легко визуализиро-

вать пересечения сложных форм. Алгоритм обладает линейной вычислительной сложностью, так как порядок добавления элементов не имеет значения, что исключает необходимость предварительной сортировки по глубине.

Основной недостаток алгоритма – большой объём требуемой памяти. Однако, поскольку размер создаваемого изображения относительно небольшой, затраты памяти на хранение информации о каждом пикселе в этом алгоритме незначительны для современных компьютеров.

### **1.4.7 Выбор оптимального алгоритма**

Для удаления невидимых линий и поверхностей был выбран алгоритм Z-буфера. Данный алгоритм прост в своей реализации, производителен и позволяет добиться достаточной детализации синтезируемого изображения.

## **1.5 Анализ модели освещения**

Модели освещения являются основным инструментом для создания реалистичных изображений. Они описывают, как свет взаимодействует с поверхностями объектов и как это взаимодействие влияет на восприятие цвета и яркости. Модели освещения можно разделить на две основные категории:

- *Локальные* модели освещения рассматривают освещение на уровне отдельной поверхности, не учитывая влияние света от других объектов. Эти модели обычно используют упрощенные подходы для расчета освещения и являются более производительными, что делает их подходящими для реального времени;
- *Глобальные* модели освещения учитывают взаимодействие света между различными объектами в сцене, что позволяет создавать более реалистичные изображения. Эти модели обычно требуют больше вычислительных ресурсов.

### **1.5.1 Модель Ламберта**

Модель Ламберта, также известная как модель диффузного отражения, основана на предположении, что интенсивность света, отраженного от поверхности, пропорциональна косинусу угла между нормалью к поверхности и направлением на источник света. Это означает, что поверхность будет выглядеть

ярче, когда она освещается под прямым углом и темнее, когда свет падает под острым углом.

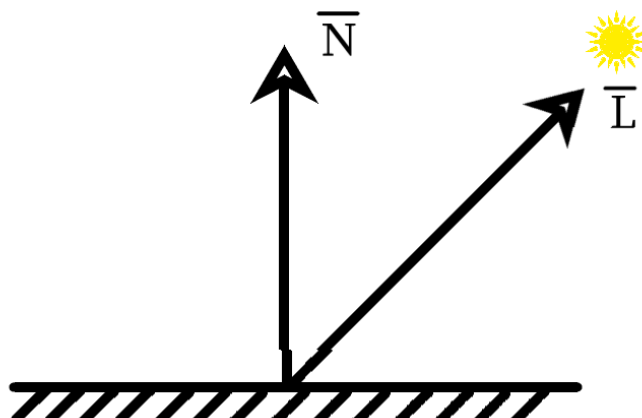


Рисунок 1.15 — Модель Ламберта

При этом положение наблюдателя не влияет на восприятие, поскольку свет, отраженный диффузно, распределяется равномерно во всех направлениях.

Формула для расчета диффузного освещения в модели Ламберта выглядит следующим образом:

$$I = I_0 \cdot k_d \cdot \cos(\vec{L}, \vec{N}) = I_0 \cdot k_d \cdot (\vec{L} \cdot \vec{N}) \quad (1.4)$$

где  $I$  – результирующая интенсивность света в точке,  $I_0$  – интенсивность источника света,  $k_d$  – коэффициент диффузного освещения,  $\vec{L}$  – вектор от точки до источника и  $\vec{N}$  – нормаль в точке.

### 1.5.2 Модель Фонга

Модель Фонга является классической моделью освещения, которая сочетает в себе диффузную составляющую и зеркальную составляющую. Это позволяет не только обеспечить равномерное освещение, но и создавать блики на материале. Положение блика на объекте, освещенном по модели Фонга, определяется законом равенства углов падения и отражения. Если наблюдатель находится близко к углу отражения, яркость данной точки увеличивается.

Падающий и отраженный лучи расположены в одной плоскости с нормалью к поверхности в точке падения, и эта нормаль делит угол между лучами пополам. Таким образом, отраженная составляющая освещенности в данной

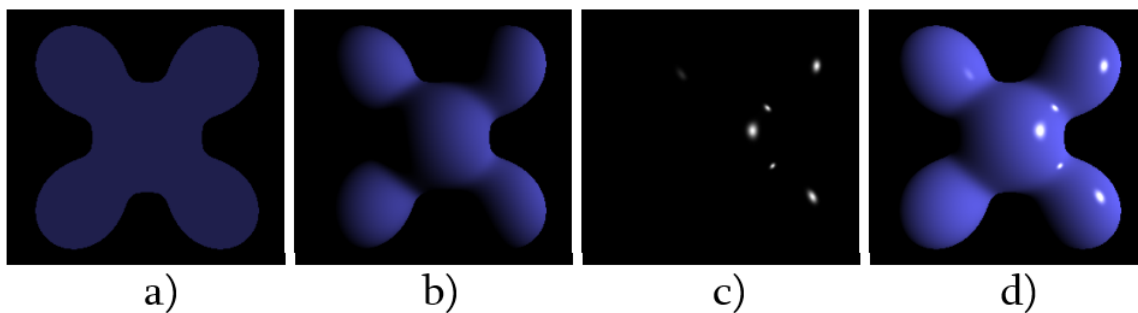


Рисунок 1.16 — Компоненты модели Фонга: рассеяная (a), диффузная (b), зеркальная (c), суммарная (d)

точке зависит от того, насколько близки направления на наблюдателя и отраженный луч.

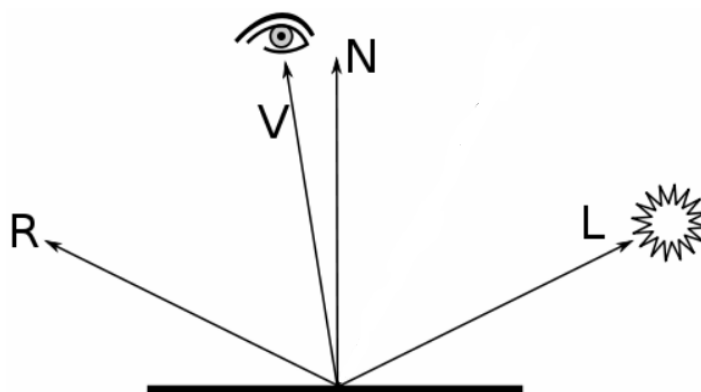


Рисунок 1.17 — Модель Фонга

Формула для расчета освещения в модели Фонга выглядит следующим образом:

$$I = I_a + I_d + I_s \quad (1.5)$$

где  $I_a$  – рассеянная составляющая,  $I_d$  – диффузная составляющая и  $I_s$  – зеркальная составляющая.

$$I_a = I_a \cdot k_a \quad (1.6)$$

где  $I_a$  – интенсивность рассеянного освещения и  $k_a$  – коэффициент рассеянного освещения.

$$I_d = I_0 \cdot k_d \cdot \cos(\vec{L}, \vec{N}) = I_0 \cdot k_d \cdot (\vec{L} \cdot \vec{N}) \quad (1.7)$$

где  $I_0$  – интенсивность источника света,  $k_d$  – коэффициент диффузного освещения,  $\vec{L}$  – вектор от точки до источника и  $\vec{N}$  – нормаль в точке.

$$I_s = I_0 \cdot k_s \cdot \cos^\alpha(\vec{R}, \vec{V}) = I_0 \cdot k_s \cdot (\vec{R} \cdot \vec{V})^\alpha \quad (1.8)$$

где  $k_s$  – коэффициент зеркального освещения,  $\vec{R}$  – вектор отраженного луча и  $\vec{V}$  – вектор от точки до наблюдателя.

$$\vec{R} = 2 \cdot (\vec{N} \cdot \vec{L}) \cdot \vec{N} - \vec{L} \quad (1.9)$$

где  $\vec{R}$  – вектор отраженного луча,  $\vec{N}$  – нормаль в точке и  $\vec{L}$  – вектор от точки до источника.

### 1.5.3 Модель Блинна-Фонга

Модель Блинна-Фонга представляет собой упрощение модели Фонга, используемое для расчета зеркального отражения света на поверхностях с меньшими вычислительными затратами. Основная идея заключается в использовании вектора полупути  $\vec{H}$ , который определяется как:

$$\vec{H} = \frac{\vec{L} + \vec{V}}{|\vec{L} + \vec{V}|} \quad (1.10)$$

где  $\vec{L}$  – вектор от точки до источника и  $\vec{V}$  – вектор от точки до наблюдателя.

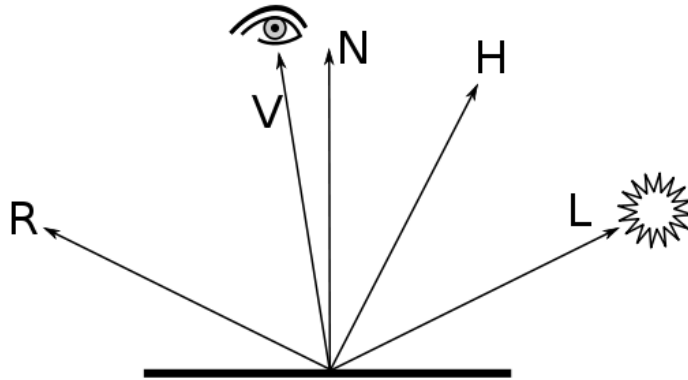


Рисунок 1.18 — Модель Блинна-Фонга

Вектор  $\vec{H}$  называется вектором полупути, поскольку, если все три вектора  $\vec{V}$ ,  $\vec{L}$  и  $\vec{N}$  лежат в одной плоскости, то угол между  $\vec{H}$  и  $\vec{N}$  составляет половину угла между  $\vec{R}$  и  $\vec{V}$ . Величину  $(\vec{R} \cdot \vec{V})^\alpha$  необходимую при расчёте зеркальной составляющей интенсивности  $I_s$  в модели Фонга (1.8) можно заменить на величину

$$(\vec{H} \cdot \vec{N})^\beta.$$

Вектор  $\vec{H}$  показывает ориентацию поверхности, на которой будет максимальное отражение света. В ряде случаев модель Блинна-Фонга требует значительно меньше вычислений, например в случае направленного бесконечно-удаленного источника.

#### 1.5.4 Выбор оптимальной модели освещения

В качестве модели освещения была выбрана модель Фонга. Данная модель учитывает как диффузное, так и зеркальное отражение света, что делает синтезируемое изображение более реалистичным и способствует возникновению бликов света.

### 1.6 Анализ алгоритмов закрашивания

Алгоритмы закрашивания определяют, как поверхности объектов будут окрашены в зависимости от их геометрии, освещения и материалов.

#### 1.6.1 Простая закрашка

Данный алгоритм представляет простейший способ закрашки. Для многогранных объектов высчитываются уровни интенсивности для каждой грани согласно закону Ламберта (1.4) и закрашка каждой грани соответствует её уровню интенсивности. Это делает процесс рендеринга простым и быстрым, поскольку не требует сложных расчетов для каждой точки на поверхности.



Рисунок 1.19 — Аппроксимированная сфера, закрашенная алгоритмом простой закрашки



Однако, простая закрашка может привести к тому, что границы между соседними гранями будут слишком резкими и заметными. Это можно исправить увеличением количества граней объекта.

### 1.6.2 Закраска Гуро

В отличие от простой закрашки, где освещение рассчитывается для всей грани, в закрашке Гуро нормали определяются в вершинах модели. Это позволяет учитывать освещение на более детальном уровне и обеспечивает более плавные переходы между цветами.

Нормали к вершинам модели вычисляются путем усреднения нормалей всех граней, которые соединяются в данной вершине. Это позволяет учитывать влияние нескольких граней на освещение в этой точке. После того как нормали вершин определены, можно вычислить освещенность каждой вершины. Закраска осуществляется вдоль рёбер многоугольников с использованием билинейной интерполяции значений интенсивности цвета в вершинах. Это позволяет создать плавный переход между цветами, что визуально сглаживает поверхность.

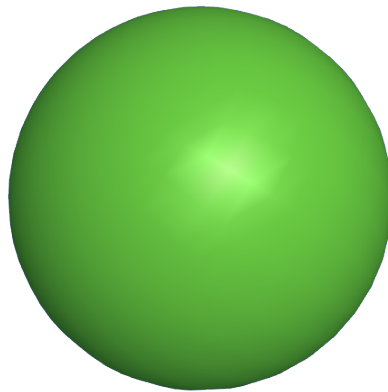


Рисунок 1.20 — Аппроксимированная сфера, закрашенная алгоритом Гуро

Одним из основных недостатков затенения по Гуро является то, что при низком уровне детализации блики на поверхности могут быть смазаны или потеряны.

### 1.6.3 Закраска Фонга

В отличие от закрашки Гуро, где цвета интерполируются между вершинами, в закрашке Фонга интерполируются нормали. Цвет, в свою очередь, рас-

считывается для каждого пикселя в отдельности. При использовании закрашки Фонга изображение получается гораздо более качественным, чем при использовании предыдущих техник, и исчезает проблема с бликами.

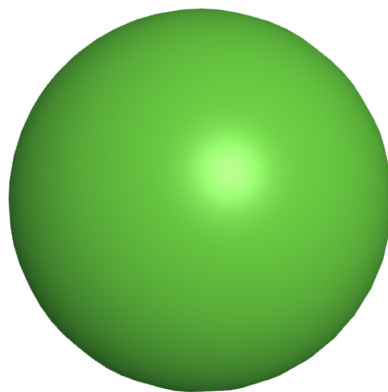


Рисунок 1.21 — Аппроксимированная сфера, закрашенная алгоритмом Фонга

Одним из основных недостатков является необходимость выполнять большое количество вычислений.

#### **1.6.4 Выбор оптимального алгоритма закрашки**

Для закрашки был выбран алгоритм Фонга. Данный алгоритм несмотря на его вычислительные затраты позволит обеспечить реалистичное освещение и четкие блики.

#### **1.7 Анализ алгоритма построения теней**

Для построения теней был выбран модифицированный алгоритм  $Z$ -буфера с добавлением теней, так как он работает с направленным источником света. Этот модифицированный алгоритм состоит из двух шагов:

- сцена строится из точки наблюдения, совпадающей с источником света. Значения  $z$  для этого вида сохраняются в отдельном теневом  $z$ -буфере, при этом значения интенсивности не учитываются.
- затем сцена строится из точки, где находится наблюдатель. При обработке каждой поверхности или многоугольника глубина в каждом пикселе сравнивается с глубиной в  $z$ -буфере наблюдателя. Если поверхность видима, координаты  $x$ ,  $y$ ,  $z$  из вида наблюдателя линейно преобразуются в координаты  $x'$ ,  $y'$ ,  $z'$  для вида из источника. Чтобы проверить, видимо

ли значение  $z'$  из положения источника, оно сравнивается со значением теневого  $z$ -буфера при  $x', y'$ . Если значение  $z'$  видимо, оно отображается в буфере кадра в точке  $x, y$  без изменений. В противном случае точка находится в тени и отображается в соответствии с правилом расчета интенсивности с учетом затенения, а значение в  $z$ -буфере наблюдателя обновляется на  $z'$ .

Этот метод создания теней требует значительных вычислительных ресурсов, но эти расчеты выполняются лишь при изменении сцены. Таким образом, если камера остается на одном месте, повторные вычисления значений в теновом  $Z$ -буфере не нужны.

## 1.8 Вывод

В аналитической части работы был проведен детальный анализ объектов сцены, способов их представления, алгоритмов удаления невидимых линий и поверхностей, моделей освещения и закрашивания, а также алгоритма построения теней. Был сделан в пользу поверхностной модели, заданной полигональной сеткой, алгоритма  $Z$ -буфера, модели освещения Фонга, закраски Фонга и модифицированного алгоритма  $Z$ -буфера с добавлением теней.

## **2 Конструкторская часть**

В этой части представляются требования к программе, алгоритм визуализации сцены, выбранные типы и структуры данных, диаграмма классов.

### **2.1 Требования к программе**

Программа должна обладать графическим интерфейсом, который позволит пользователю:

- изменять геометрические и спектральные характеристики модели многогранника;
- изменять положение модели многогранника в пространстве;
- изменять положение камеры в пространстве;
- изменять режим отображения сцены между «каркасным», «реалистичным», «световым» и «теневым», отображающими каркасы моделей, линии пересечения моделей, сцену с учётом освещения и сцену с тенями, возникающими от освещения, соответственно.

Разработанная программа должна выполнять следующие требования:

- источник света должен создаваться при запуске;
- в пространстве может быть только один источник света;
- в пространстве может быть только одна камера;
- программа обязана правильно обрабатывать некорректные вводимые данные.

### **2.2 Общий алгоритм визуализации сцены**

На рисунке 2.1 представлен алгоритм, который генерирует изображение. Он принимает на вход геометрические и спектральные параметры моделей, характеристики камеры, данные об источнике света и его спектральные свойства, а на выходе предоставляет визуализированную сцену на экране.

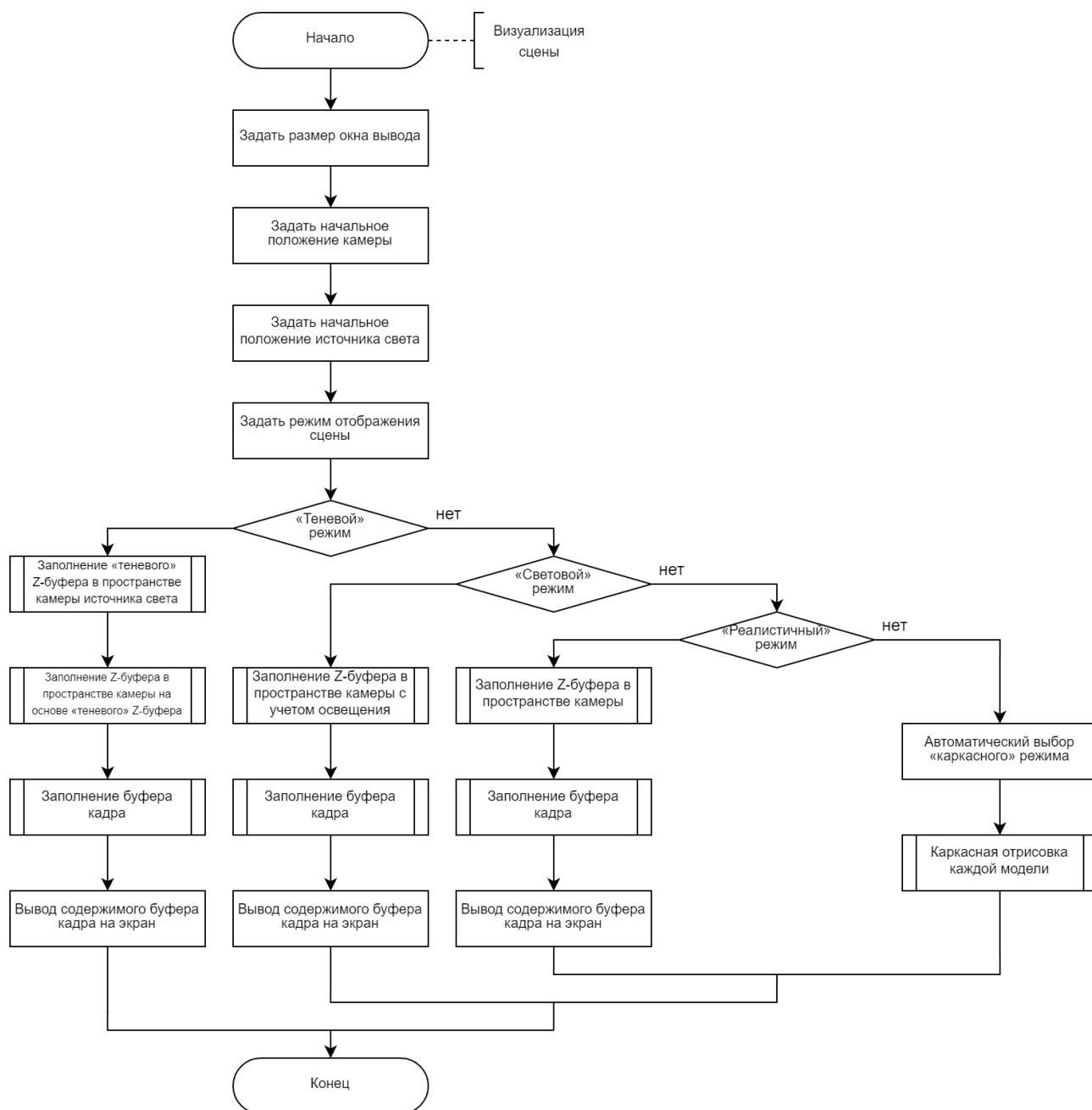


Рисунок 2.1 — Общий алгоритм визуализации сцены

Более подробный алгоритм визуализации сцены в «теневом» режиме отображения сцены при инициализированной сцене представлен на рисунке 2.2.

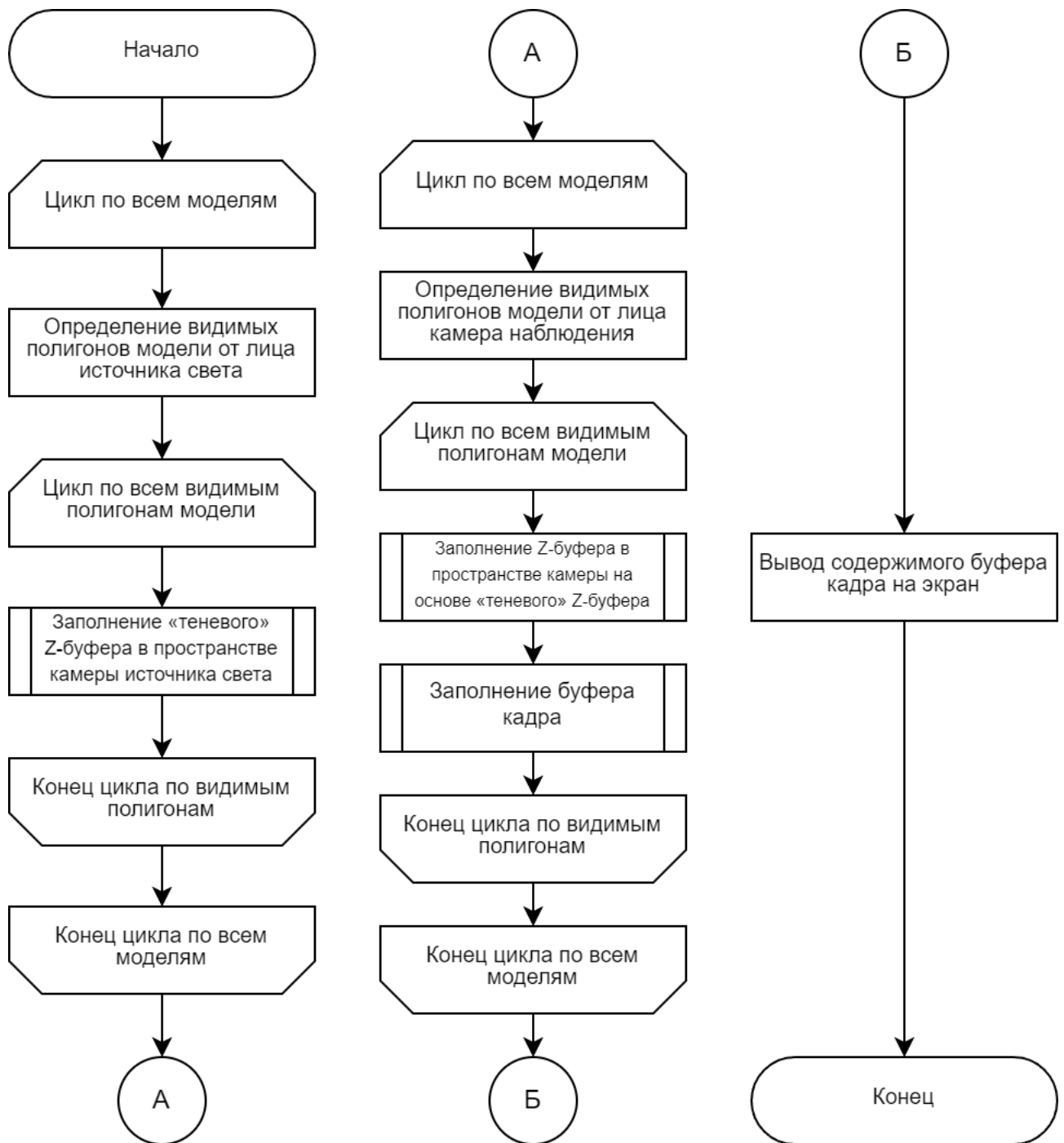


Рисунок 2.2 — Общий алгоритм визуализации сцены в «теневом» режиме при инициализированной сцене

Более подробный алгоритм визуализации сцены в «световом», «реалистичном» и «каркасном» режимах отображения сцены при инициализированной сцене представлены на рисунке 2.3.

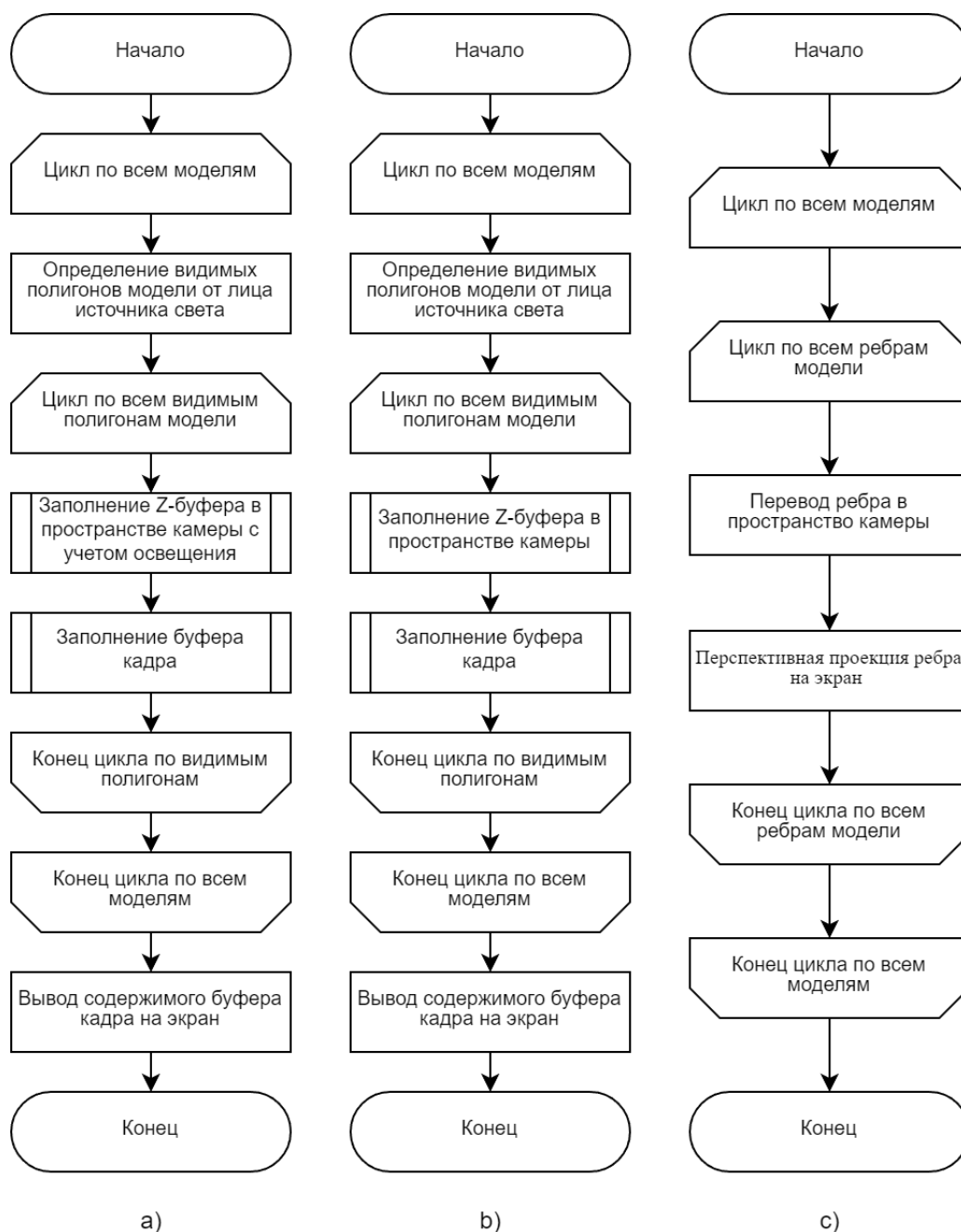


Рисунок 2.3 — Общий алгоритм визуализации сцены в иных режимах при инициализированной сцене: «световой» (а), «реалистичный» (b), «каркасный» (с)

## 2.3 Аффинные преобразования

Для изменения положения модели в пространстве можно использовать следующие преобразования: перемещение и поворот. Каждое из этих преобразований может быть представлено в виде матрицы:

- 1) перемещение вдоль координатных осей  $O_X$ ,  $O_Y$  и  $O_Z$  соответственно на величины  $d_x$ ,  $d_y$ ,  $d_z$ :

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ d_x & d_y & d_z & 1 \end{pmatrix} \quad (2.1)$$

- 2) поворот вокруг координатных осей  $O_X$ ,  $O_Y$  и  $O_Z$  на угол  $\theta$ :

$$R = R_X \cdot R_Y \cdot R_Z \quad (2.2)$$

где  $R$  – матрица поворота,  $R_X$  матрица поворота вокруг оси  $O_X$  на угол  $\theta$ ,  $R_Y$  – матрица поворота вокруг оси  $O_Y$  на угол  $\theta$ ,  $R_Z$  – матрица поворота вокруг оси  $O_Z$  на угол  $\theta$ .

- поворот вокруг оси  $O_X$  на угол  $\theta$ :

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.3)$$

- поворот вокруг оси  $O_Y$  на угол  $\theta$ :

$$\begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.4)$$



– поворот вокруг оси  $O_Z$  на угол  $\theta$ :

$$\begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 & 0 \\ -\sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.5)$$

## 2.4 Камера

Камера отвечает за захват и отображение части трёхмерной сцены на двумерную поверхность. Она определяет, как объекты будут видны зрителю, а также управляет параметрами визуализации, такими как угол обзора.

### 2.4.1 Пирамида видимого пространства

Пирамида видимого пространства (*англ. viewing frustum*) представляет собой объем, в пределах которого объекты могут быть видимы камерой. Этот объем имеет форму усеченной пирамиды (рис. 2.4) и определяется несколькими ключевыми параметрами:

- ближняя плоскость: плоскость, находящаяся на определенном расстоянии от камеры, перед которой объекты начинают отображаться. Объекты, находящиеся ближе, не будут видны;
- дальняя плоскость: плоскость, за которой объекты не отображаются;
- поле зрения (*англ. field of view*): угол, под которым камера охватывает сцену. Поле зрения влияет на то, насколько широко камера может видеть, и определяет степень искажения объектов в зависимости от их расстояния до камеры.

Пирамида видимого пространства позволяет игнорировать объекты вне этого объема, что увеличивает скорость рендеринга.

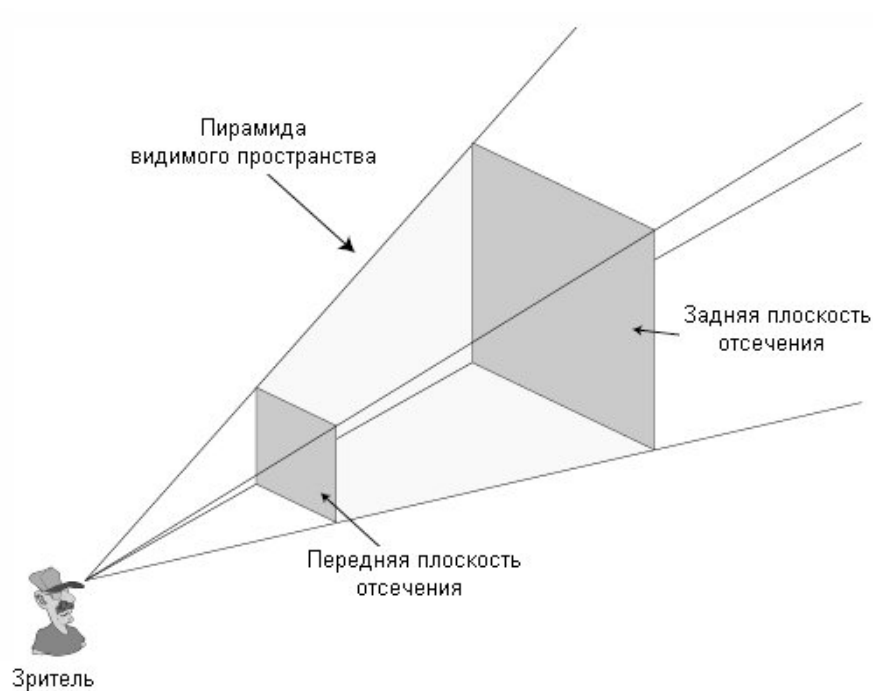


Рисунок 2.4 — Пирамида видимого пространства

### 2.4.2 Пространство модели

Пространство модели – это локальная система координат, в которой располагается конкретный объект сцены. В этом пространстве объект имеет координаты, которые определяют его положение, ориентацию и масштаб относительно своей собственной системы координат.

Пусть вершина  $V_L$  имеет однородные координаты  $(x_L, y_L, z_L, 1)$  в пространстве модели, тогда её можно описать с помощью матрицы:

$$V_L = \begin{pmatrix} x_L \\ y_L \\ z_L \\ 1 \end{pmatrix} \quad (2.6)$$

### 2.4.3 Мировое пространство

Мировое пространство – это система координат, в которой расположены все объекты сцены. В этом пространстве каждое тело имеет свои координаты, определяющие его положение, ориентацию и масштаб. Мировое пространство позволяет организовать объекты в сцене.

Для перевода модели из её пространства в мировое необходима матрица перевода. Для её формирования необходимо составить матрицу перемещения модели относительно начала мировой системы координат и матрицу поворота модели относительно её центра.

$$W = M_W \cdot R_W \quad (2.7)$$

где  $W$  – матрица перевода модели из её пространства в мировое,  $M_W$  – матрица перемещения,  $R_W$  – матрица поворота.

Чтобы перевести вершину  $V_L$  (2.6) из её локального пространства в мировое, её нужно умножить на матрицу перевода  $W$ :

$$V_W = W \cdot V_L \quad (2.8)$$

где  $V_W$  – матричная запись вершины  $V_L$ , переведённой из её локального пространства в мировое и имеющей однородные координаты  $(x_W, y_W, z_W, w_W)$ ,  $W$  – матрица перевода модели из её пространства в мировое.

#### 2.4.4 Пространство камеры

Пространство камеры — это система координат, в которой камера воспринимает объекты. Основные компоненты пространства камеры включают:

- положение камеры  $P$ : определяет, где находится камера в мировом пространстве;
- ориентация камеры  $\vec{D}$ : направление взгляда камеры, задаваемое вектором;
- система координат  $\vec{D}, \vec{U}, \vec{R}$ : устанавливает оси  $O_X, O_Y$  и  $O_Z$  для камеры, где ось  $\vec{D}$  направлена вперед (ампликата), ось  $\vec{U}$  – вверх (ордината), а ось  $\vec{R}$  – вправо (абсцисса).

$$\vec{R} = \vec{D} \times O_Y \quad (2.9)$$

$$\vec{U} = \vec{D} \times \vec{R} \quad (2.10)$$

Для перевода модели из мирового пространства в пространство камеры необходима матрица перевода. Для её формирования необходимо составить

матрицу перемещения камеры относительно начала мировой системы координат и матрицу поворота камеры в мировом пространстве.

$$M_C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -P_x & -P_y & -P_z & 1 \end{pmatrix} \quad (2.11)$$

где  $M_C$  – матрица перемещения камеры относительно начала мировой системы координат,  $P$  – положение камеры в мировом пространстве.

$$R_C = \begin{pmatrix} \vec{R}_x & \vec{U}_x & \vec{D}_x & 0 \\ \vec{R}_y & \vec{U}_y & \vec{D}_y & 0 \\ \vec{R}_z & \vec{U}_z & \vec{D}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.12)$$

где  $R_C$  – матрица поворота камеры в мировом пространстве,  $\vec{R}$  – абсцисса системы координат камеры,  $\vec{U}$  – ордината системы координат камеры,  $\vec{D}$  – направление взгляда камеры (ампликата системы координат камеры).

$$C = M_C \cdot R_C \quad (2.13)$$

где  $C$  – матрица перевода модели из мирового пространства в пространство камеры,  $M_C$  – матрица перемещения камеры относительно начала мировой системы координат,  $R_C$  – матрица поворота камеры в мировом пространстве.

Чтобы перевести вершину  $V_W$  (2.8) из мирового пространства в пространство камеры, её нужно умножить на матрицу перевода:

$$V_C = C \cdot V_W \quad (2.14)$$

где  $V_C$  – матричная запись вершины  $V_W$ , переведённой из мирового пространства в пространство камеры и имеющей однородные координаты  $(x_C, y_C, z_C, w_C)$ ,  $C$  – матрица перевода модели из мирового пространства в пространство камеры.

### 2.4.5 Пространство отсечения

Пространство отсечения – это трехмерное пространство в виде куба, в котором все вершины объектов представлены в координатах, нормализованных в диапазоне от -1 до 1 для каждой из осей  $O_X$ ,  $O_Y$  и  $O_Z$ . Если вершина после её перевода в пространство отсечения имеет какую-либо координату, значение которой не входит в промежуток от -1 до 1, то рендеринг этой вершины прекращается.

Для перевода модели из пространства камеры в пространство отсечения необходима матрица *перспективной проекции*. Для её формирования нужны значения ширины и высоты экрана, вертикального угла обзора, расстояний до ближней и дальней плоскостей пирамиды видимого пространства.

$$P = \begin{pmatrix} \frac{1}{\operatorname{tg}(\frac{\gamma}{2}) \cdot \frac{W}{H}} & 0 & 0 & 0 \\ 0 & \frac{1}{\operatorname{tg}(\frac{\gamma}{2})} & 0 & 0 \\ 0 & 0 & -\frac{F+N}{F-N} & -2\frac{F \cdot N}{F-N} \\ 0 & 0 & -1 & 0 \end{pmatrix} \quad (2.15)$$

где  $P$  – матрица перевода из пространства камеры в пространство отсечения,  $\gamma$  – вертикальный угол обзора,  $W$  – ширина экрана,  $H$  – высота экрана,  $F$  – расстояние до дальней плоскости пирамиды видимого пространства,  $N$  – расстояние до ближней плоскости пирамиды видимого пространства.

Запись матрицы  $P$  (2.15) можно упростить. Пусть  $Z_y = \frac{1}{\operatorname{tg}(\frac{\gamma}{2})}$ ,  $R = \frac{W}{H}$ ,  $Z_x = \frac{Z_y}{R}$ , тогда:

$$P = \begin{pmatrix} Z_x & 0 & 0 & 0 \\ 0 & Z_y & 0 & 0 \\ 0 & 0 & -\frac{F+N}{F-N} & -2\frac{F \cdot N}{F-N} \\ 0 & 0 & -1 & 0 \end{pmatrix} \quad (2.16)$$

Чтобы перевести вершину  $V_C$  (2.14) из пространства камеры в пространство отсечения, её нужно умножить на матрицу перевода:

$$V_P = P \cdot V_C \quad (2.17)$$

где  $V_P$  – матричная запись вершины  $V_C$ , переведенной из пространства камеры

в пространство отсечения и имеющей однородные координаты  $(x_P, y_P, z_P, w_P)$ ,  $P$  – матрица перевода из пространства камеры в пространство отсечения.

Чтобы проверить принадлежность вершины  $V_P$  (2.17) пространству отсечения, её однородные координаты нужно нормализовать и проверить принадлежность их значений промежутку от -1 до 1:

$$\begin{cases} -1 \leq x_{NP} \leq 1 \\ -1 \leq y_{NP} \leq 1 \\ -1 \leq z_{NP} \leq 1 \end{cases} \quad (2.18)$$

где  $x_{NP} = \frac{x_P}{w_P}$  – нормализованная координата  $x_P$ ,  $y_{NP} = \frac{y_P}{w_P}$  – нормализованная координата  $y_P$ ,  $z_{NP} = \frac{z_P}{w_P}$  – нормализованная координата  $z_P$ .

## 2.4.6 Пространство экрана

Пространство экрана – это координатная система, в которой трехмерные объекты сцены отображаются как двумерные изображения на экране. Данная координатная система определяется двумя осями  $O_X$  и  $O_Y$ , направленными вправо и вниз соответственно, и имеет фиксированные размеры, которые зависят от разрешения экрана.

Для перевода вершины  $V_P$  (2.17) из пространства отсечения в пространство экрана, её вершины необходимо предварительно нормализовать. Пусть  $V_{NP}$  – нормализованная вершина  $V_P$ , имеющая однородные координаты  $(x_{NP}, y_{NP}, z_{NP}, 1)$ , тогда:

$$\begin{aligned} D_x &= (x_{NP} \cdot 0.5 + 0.5) \cdot W \\ D_y &= (y_{NP} \cdot 0.5 + 0.5) \cdot H \end{aligned} \quad (2.19)$$

где  $D$  – вершина  $V_P$ , переведенная из пространства отсечения в пространство экрана и имеющая координаты  $(D_x, D_y)$ ,  $W$  – ширина экрана,  $H$  – высота экрана.

## 2.4.7 Алгоритм получения изображения от лица камеры

На рисунке 2.5 представлен алгоритм получения изображения модели на экране от лица камеры. Он принимает на вход вершины модели в её локальном пространстве, а на выходе предоставляет изображение вершин модели от лица камеры на экране.

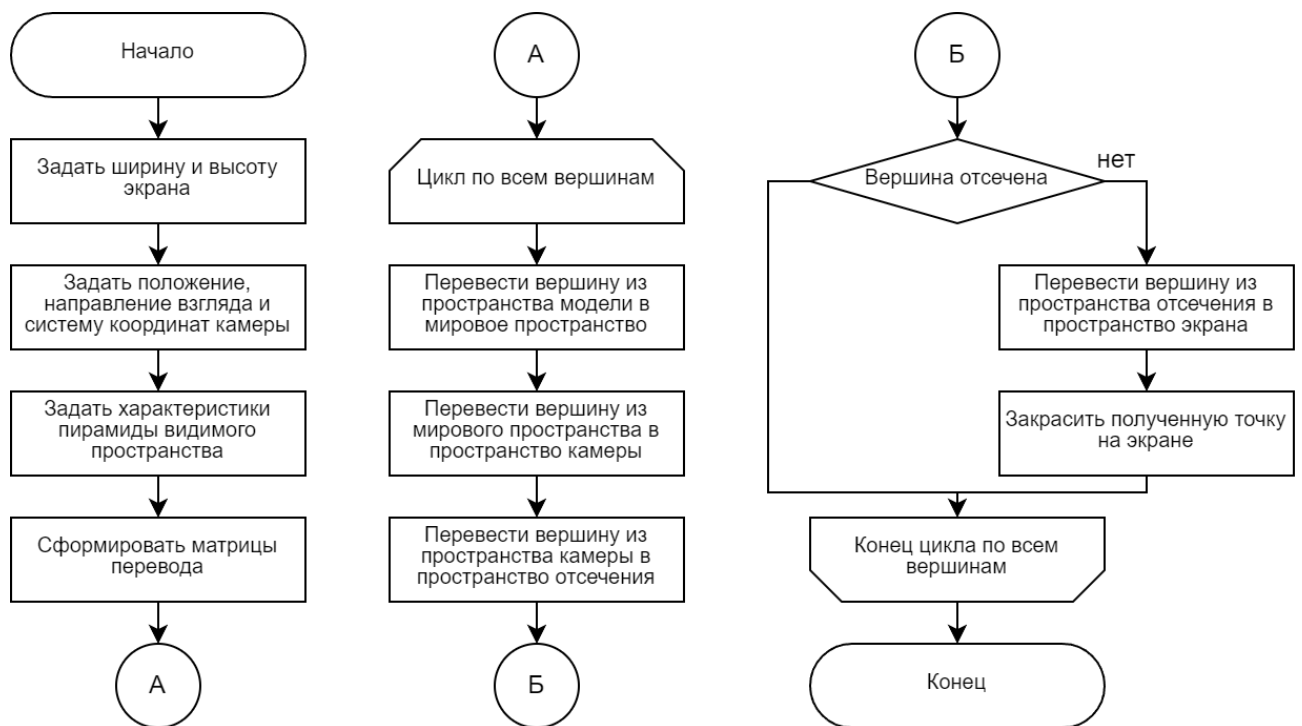


Рисунок 2.5 — Алгоритм получения изображения модели на экране от лица камеры

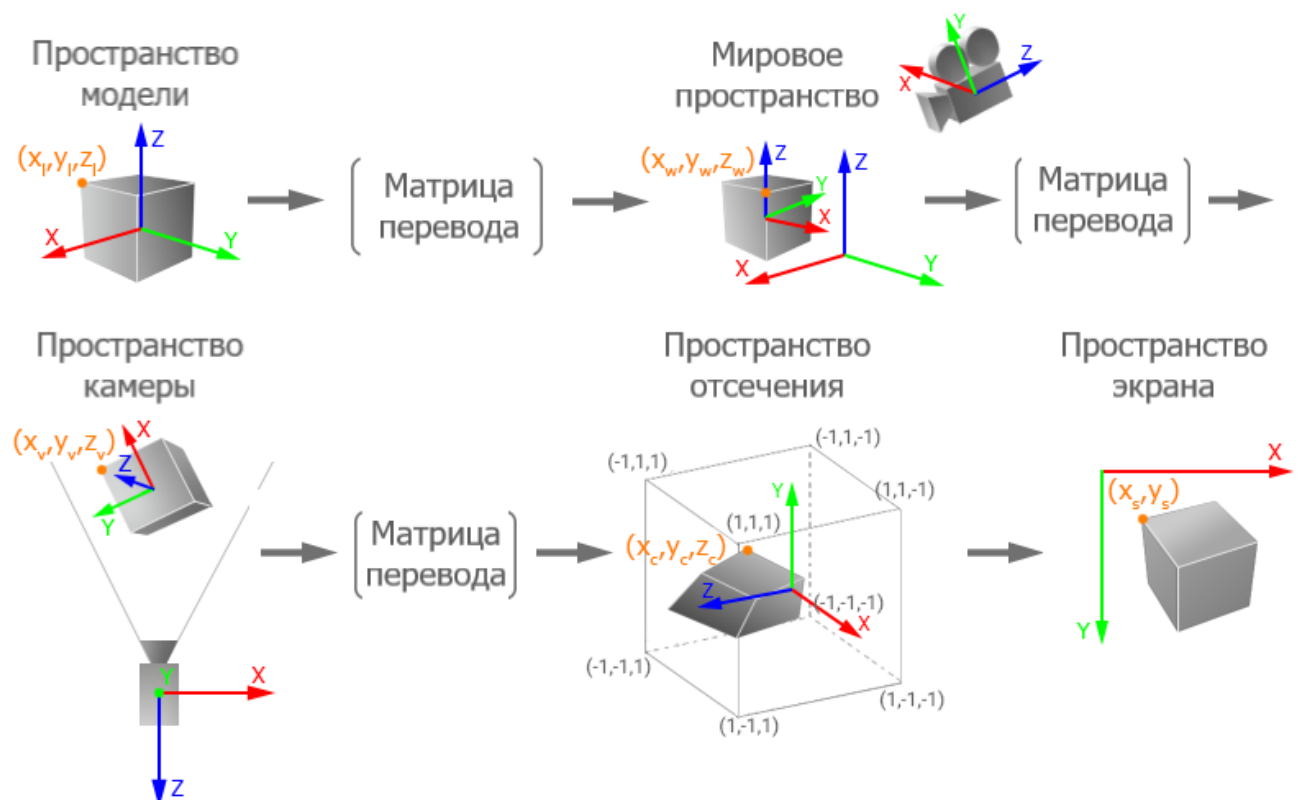


Рисунок 2.6 — Последовательный перевод модели из одного пространства в другое

## 2.5 Невидимые грани

Отбрасывание граней, которые не видны с точки зрения камеры, значительно ускоряет процесс рендеринга изображений. Это связано с тем, что невидимые полигоны не требуют обработки, что позволяет сэкономить ресурсы. Чтобы определить, видима ли грань, используется следующая формула:

$$(\vec{N}, \vec{V}) = \begin{cases} \geq 0, & \text{если грань видима} \\ < 0, & \text{если грань невидима} \end{cases} \quad (2.20)$$

где  $\vec{N}$  – вектор, перпендикулярный поверхности грани и направленный наружу от модели,  $\vec{V}$  – вектор, указывающий от камеры к любой точке на грани.

## 2.6 Алгоритм Z-буфера

Для применения алгоритма Z-буфера к модели необходимо определить все точки, принадлежащие полигону. Поскольку количество точек на полигоне бесконечно, под поиском всех точек подразумевается создание конечного набора точек, который адекватно аппроксимирует полигон при выводе на экран, обеспечивая непрерывность и избегая эффекта «сетчатости».

Для эффективного рендеринга полигона необходимо предварительно определить его ограничивающую прямую призму. Это позволит установить конкретную область в трехмерном пространстве, в которой расположен полигон.

Пусть  $P$  – полигон, заданный вершинами  $(P_1(x_1, y_1, z_1), P_2(x_2, y_2, z_2), P_3(x_3, y_3, z_3), P_4(x_4, y_4, z_4))$ ,  $B$  – ограничивающая прямая призма, заданная диагональю  $D$  с вершинами  $(D_1(D_{1x}, D_{1y}, D_{1z}), D_2(D_{2x}, D_{2y}, D_{2z}))$ ,  $B_1$  – нижнее основание призмы  $B$ , заданное вершинами  $(A, B, C, D)$ ,  $B_2$  – верхнее основание призмы  $B$ , заданное вершинами  $(A_1, B_1, C_1, D_1)$ .

$$\begin{aligned} D_{1x} &= \min(x_1, x_2, x_3, x_4) \\ D_{1y} &= \min(y_1, y_2, y_3, y_4) \\ D_{1z} &= \min(z_1, z_2, z_3, z_4) \end{aligned} \quad (2.21)$$



$$\begin{aligned}
D_{2_x} &= \max(x_1, x_2, x_3, x_4) \\
D_{2_y} &= \max(y_1, y_2, y_3, y_4) \\
D_{2_z} &= \max(z_1, z_2, z_3, z_4)
\end{aligned}
\tag{2.22}$$

Имея координаты вершин диагонали  $D$ , можно определить координаты всех вершин ограничивающей призмы  $B$ :

$$\begin{aligned}
&A(D_{1_x}, D_{1_y}, D_{1_z}), B(D_{2_x}, D_{1_y}, D_{1_z}), \\
&C(D_{2_x}, D_{2_y}, D_{1_z}), D(D_{1_x}, D_{2_y}, D_{1_z})
\end{aligned}
\tag{2.23}$$

$$\begin{aligned}
&A_1(D_{1_x}, D_{1_y}, D_{2_z}), B_1(D_{1_x}, D_{2_y}, D_{2_z}), \\
&C_1(D_{2_x}, D_{2_y}, D_{2_z}), D_1(D_{2_x}, D_{1_y}, D_{2_z})
\end{aligned}
\tag{2.24}$$

Для формирования набора точек, принадлежащих полигону, можно проанализировать множество внутренних точек ограничивающей призмы с заданной точностью и проверить каждую из них на принадлежность полигону. Однако, более эффективным подходом будет анализ множества точек с заданной точностью, полученных из проекции ограничивающей призмы на плоскость  $XY$ . Для каждой из этих точек можно вычислить координату  $z$  с использованием уравнения плоскости (1.1), которой принадлежит полигон. Это значительно сократит количество рассматриваемых трехмерных точек и позволит избежать необходимости проверки каждой точки на принадлежность плоскости полигона.

Пусть  $P$  – полигон, заданный  $n$  вершинами  $(P_1, P_2, \dots, P_n)$ ,  $A$  – точка в плоскости полигона. Для определения принадлежности точки  $A$  полигону  $P$  можно использовать метод, основанный на проверке видимости точки относительно всех сторон полигона. Данный метод осуществляется в два этапа:

- 1) определение внутренней нормали к каждой из сторон: для определения внутренней нормали  $\vec{N}_i$  к стороне  $P_i P_{i+1}$  предварительно вычисляется нормаль  $\vec{V}$  к полигону.

$$\vec{V} = \overline{P_{i-1}P_i} \times \overline{P_i P_{i+1}}
\tag{2.25}$$

$$\vec{N}_i = \overline{P_i P_{i+1}} \times \vec{V}
\tag{2.26}$$

- 2) проверка видимости относительно каждой стороны полигона: для каж-

дой стороны  $P_i P_{i+1}$  вычисляется вектор  $W_i = \overline{A f_i}$ , где  $f_i$  – произвольная точка на стороне  $P_i P_{i+1}$ . Затем вычисляется скалярное произведение  $\vec{N}_i \cdot \vec{W}_i$ . Если условие  $\vec{N}_i \cdot \vec{W}_i > 0$ , то точка  $A$  находится на видимой стороне относительно текущей стороны. Если для всех сторон полигона это условие выполняется, то точка принадлежит полигону.

На рисунке 2.7 представлен алгоритм получения множества точек полигона. Он принимает на вход вершины полигона, а на выходе предоставляет массив трехмерных точек, принадлежащих полигону.

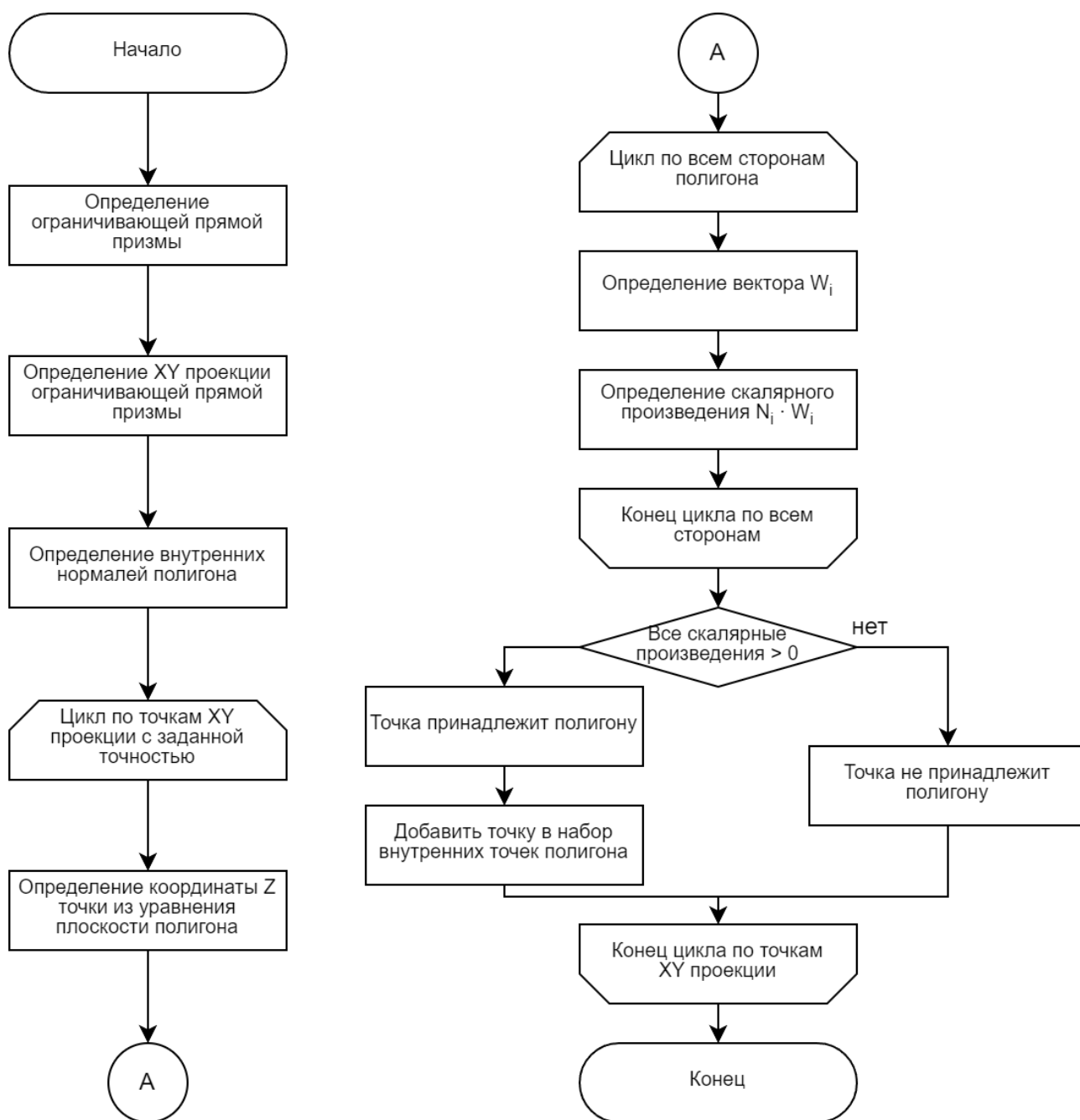


Рисунок 2.7 — Алгоритм получения множества точек полигона

На рисунке 2.8 представлен алгоритм Z-буфера. Он принимает на вход геометрические параметры моделей, характеристики камеры, а на выход предоставляет изображение множества моделей сцены от лица камеры на экране.

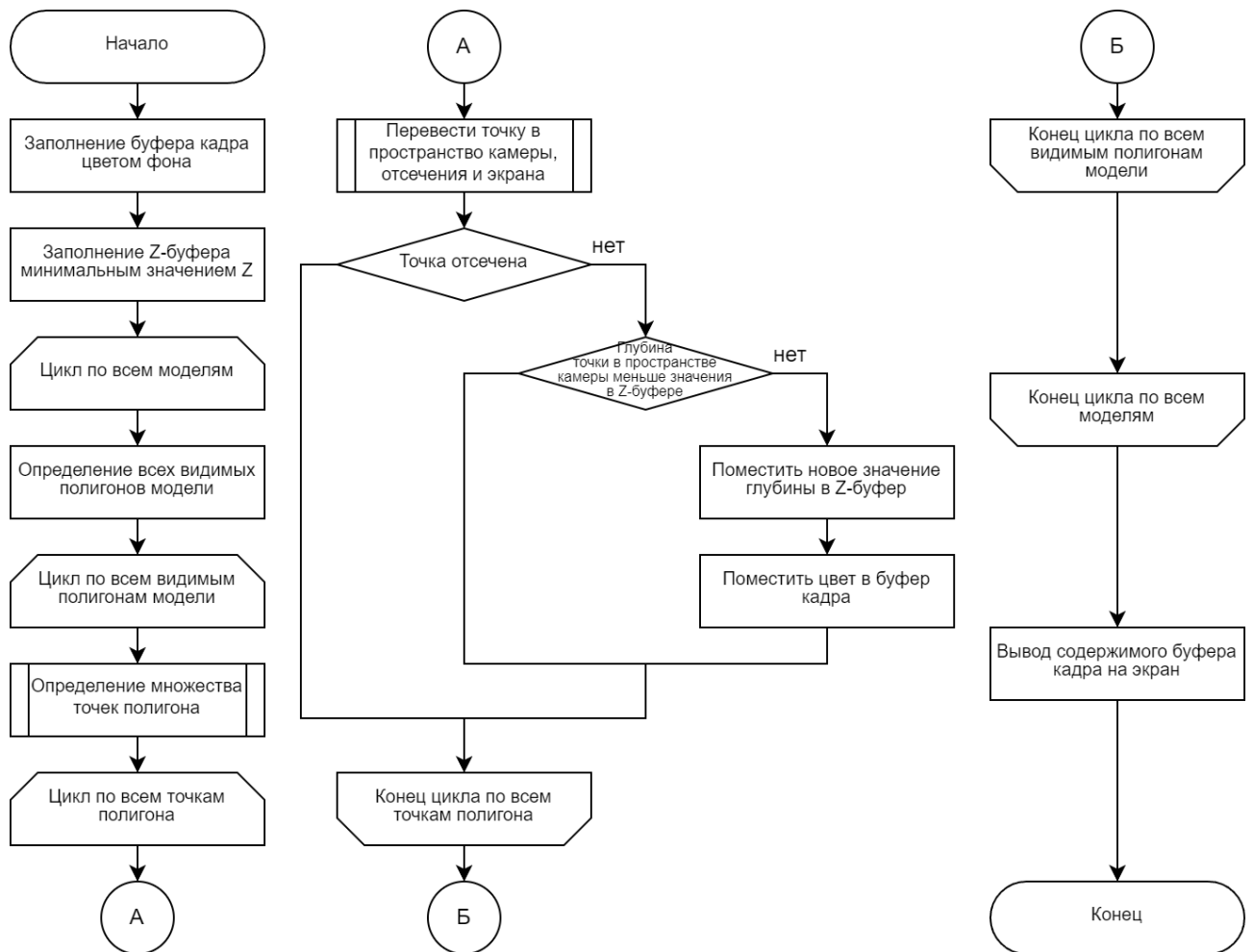


Рисунок 2.8 — Алгоритм Z-буфера

## 2.7 Выбор типов и структур данных

Используемые в программе структуры данных:

1) модель многогранника, включающая в себя:

- массив вершин;
- массив ребер;
- массив полигонов;
- геометрические характеристики;
- спектральные характеристики;
- цвет граней.

2) источник света, включающий в себя:

- положение в пространстве;
  - направление потока света.
- 3) сцена, включающая в себя:
- массив моделей;
  - источник света.
- 4) камера, включающая в себя:
- положение в пространстве;
  - направление просмотра;
  - система координат, задаваемая тремя ортогональными векторами;
  - характеристики пирамиды видимого пространства.
- 5) экран, включающий в себя:
- сцену;
  - камеру;
  - значения сторон экрана.

## **2.8 Диаграмма классов**

На рисунке 2.9 представлена диаграмма классов.

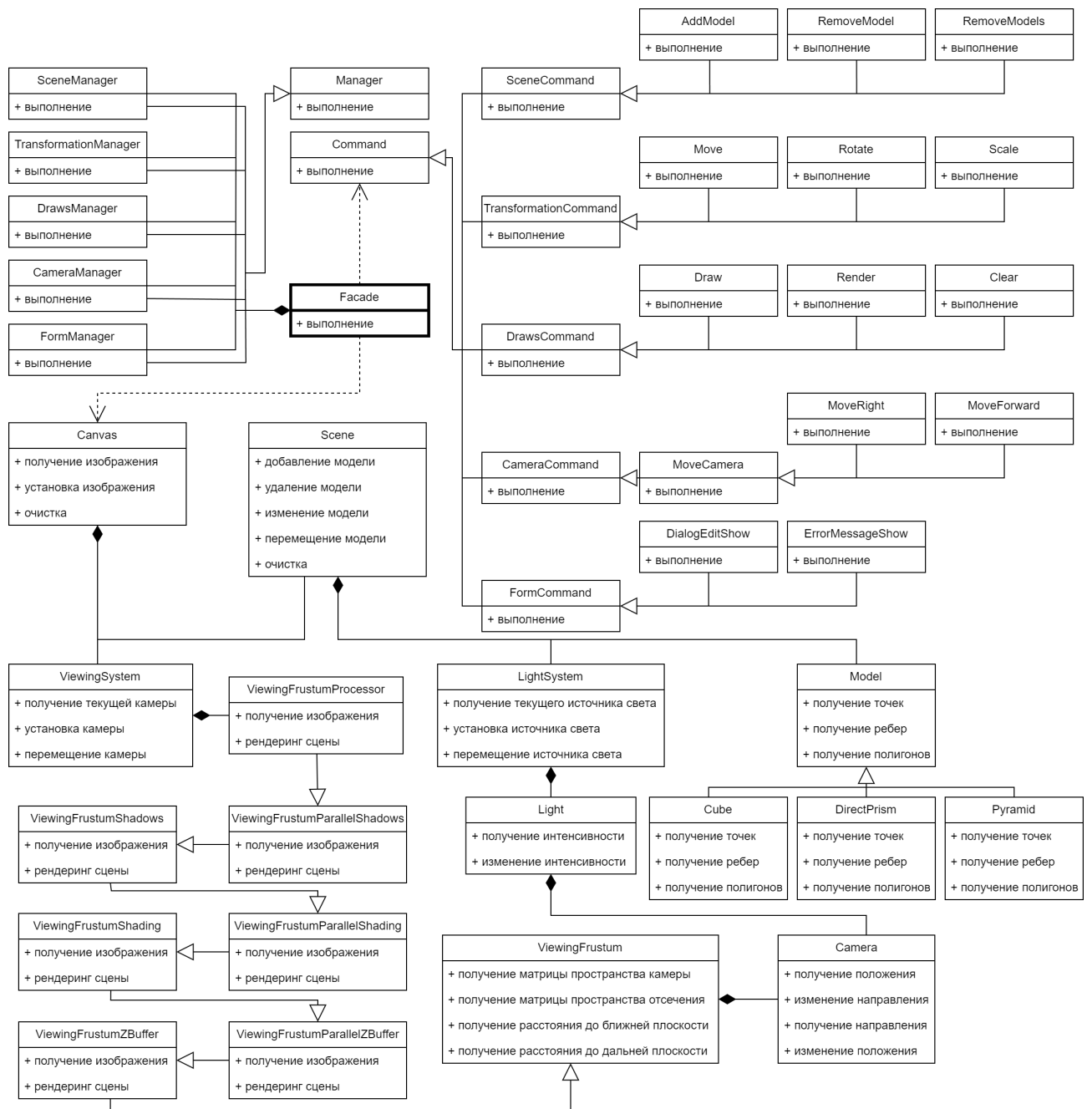


Рисунок 2.9 — Диаграмма классов

## 2.9 Вывод

В конструкторской части работы были представлены требования к программе, алгоритм визуализации сцены, выбранные типы и структуры данных, диаграмма классов.

### 3 Технологическая часть

В данной части представляются выбор средств реализации и исходный код программы, описываются организация классов в программе и её интерфейс.

#### 3.1 Средства реализации

В качестве языка программирования для реализации данной курсовой работы был выбран *C#* по следующим причинам:

- в стандартной библиотеке *C#* присутствуют необходимые структуры и классы, выбранные по результатам проектирования;
- поддержка объектно-ориентированного программирования;
- поддержка механизма многопоточности.

Для измерения процессорного времени выполнения кода был выбран класс *Stopwatch*, который находится в пространстве имен *System.Diagnostics*.

#### 3.2 Структура программы

Разработанная программа состоит из следующих классов:

1) управляющий класс:

- *Program* – точка входа в программу.

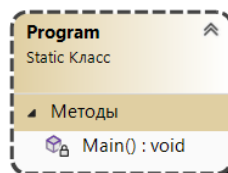


Рисунок 3.1 — Управляющий класс

2) классы интерфейса:

- *Form1* – главное окно программы;
- *DialogEdit* – окно редактирования модели;
- *ErrorMessage* – окно, содержащее сообщение об ошибке.

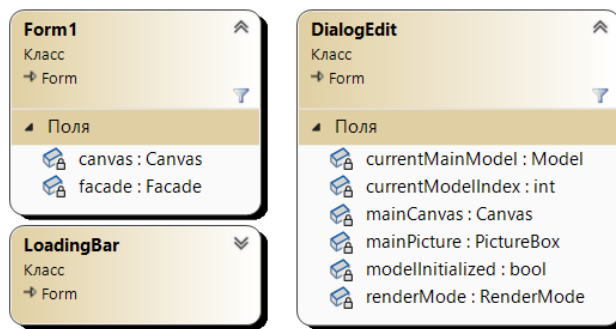


Рисунок 3.2 — Классы интерфейса

3) классы инкапсуляции действий:

- *Command* – базовый класс команды для выполнения;
- *SceneCommand* – базовый класс команды для обработки сцены;
- *CameraCommand* – базовый класс команды для обработки камеры;
- *DrawsCommand* – базовый класс команды для обработки экрана;
- *TransformationCommand* – базовый класс команды для преобразований объектов;
- *FormCommand* – базовый класс команды для обработки интерфейса.

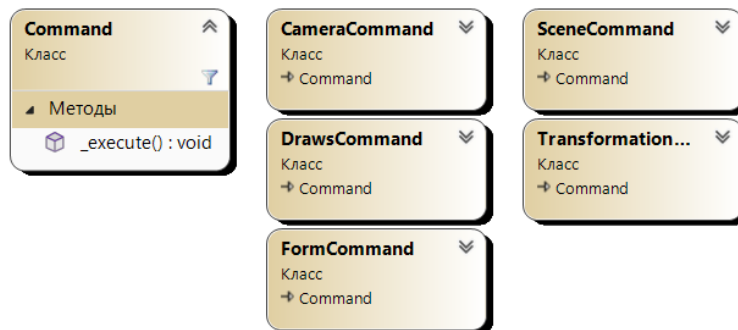


Рисунок 3.3 — Классы инкапсуляции действий

4) классы обработки команд:

- *Facade* – единый интерфейс для обработки команд;
- *SceneManager* – обработчик команд, работающих со сценой;
- *DrawManager* – обработчик команд, работающих с экраном;
- *TransformationManager* – обработчик команд, преобразовывающих объекты;
- *FormManager* – обработчик команд, работающих с интерфейсом.

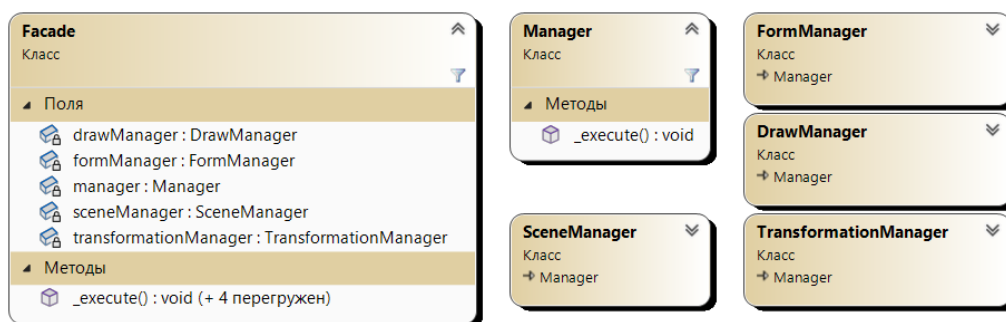


Рисунок 3.4 — Классы обработки команд

##### 5) классы взаимодействия со сценой:

- *Canvas* – менеджер изображения на экране;
- *Scene* – менеджер трехмерной сцены;
- *ViewingSystem* – менеджер просмотра сцены.

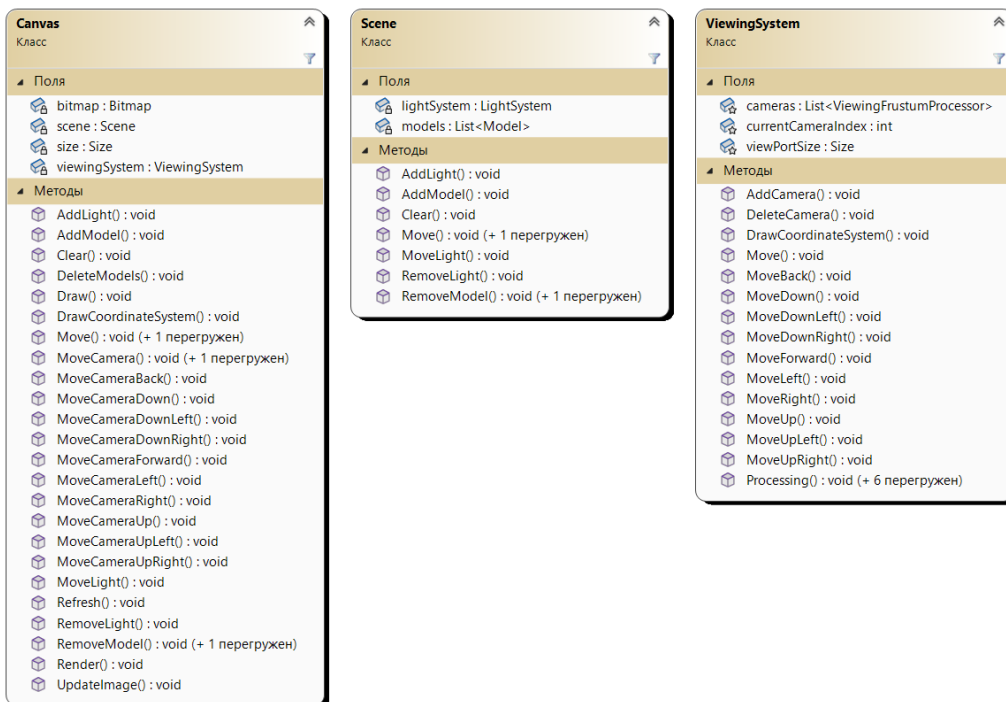


Рисунок 3.5 — Классы взаимодействия со сценой

##### 6) классы представления света:

- *LightSystem* – менеджер источников света;
- *Light* – источник света.



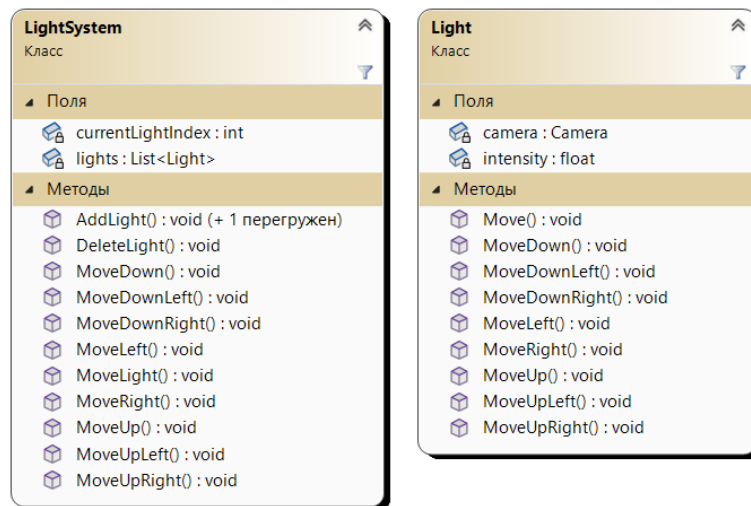


Рисунок 3.6 — Классы представления света

7) классы представления моделей:

- *Model* – базовый класс многогранной модели;
- *Cube* – куб;
- *DirectPrism* – прямая призма;
- *Pyramid* – треугольная пирамида.

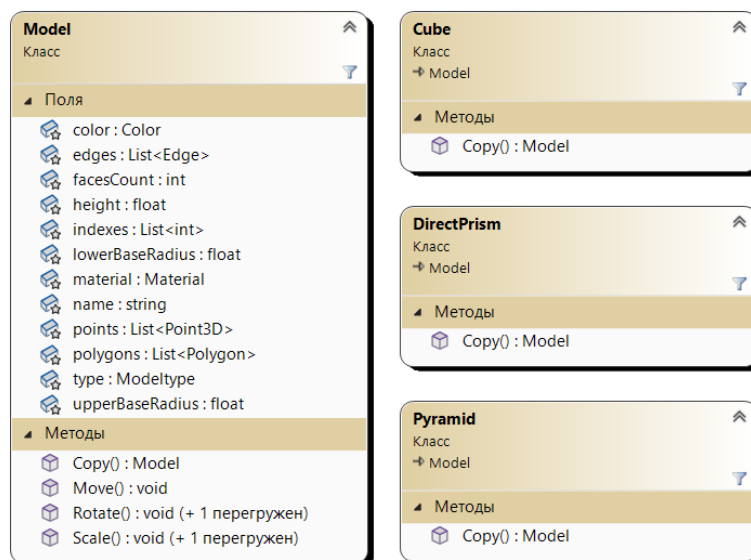


Рисунок 3.7 — Классы представления моделей

8) классы представления камеры:

- *Camera* – камера;
- *ViewingFrustum* – камера, учитывающая видимое пространство и визуализирующая сцену с использованием перспективных преобразований;

- *ViewingFrustumZBuffer* – камера, учитывающая видимое пространство, визуализирующая сцену с использованием перспективных преобразований и удалением невидимых линий и поверхностей;
- *ViewingFrustumParallelZBuffer* – камера *ViewingFrustumZBuffer*, реализованная с использованием параллельных потоков;
- *ViewingFrustumPhongShading* – камера, учитывающая видимое пространство, визуализирующая сцену с использованием перспективных преобразований, удалением невидимых линий и поверхностей и расчетом освещенности объектов;
- *ViewingFrustumParallelPhongShading* – камера *ViewingFrustumPhongShading*, реализованная с использованием параллельных потоков;
- *ViewingFrustumShadows* – камера, учитывающая видимое пространство, визуализирующая сцену с использованием перспективных преобразований, удалением невидимых линий и поверхностей, расчетом освещенности объектов и возникающих теней;
- *ViewingFrustumParallelShadows* – камера *ViewingFrustumShadows*, реализованная с использованием параллельных потоков;
- *ViewingFrustumProcessor* – камера с многорежимной визуализацией сцены.

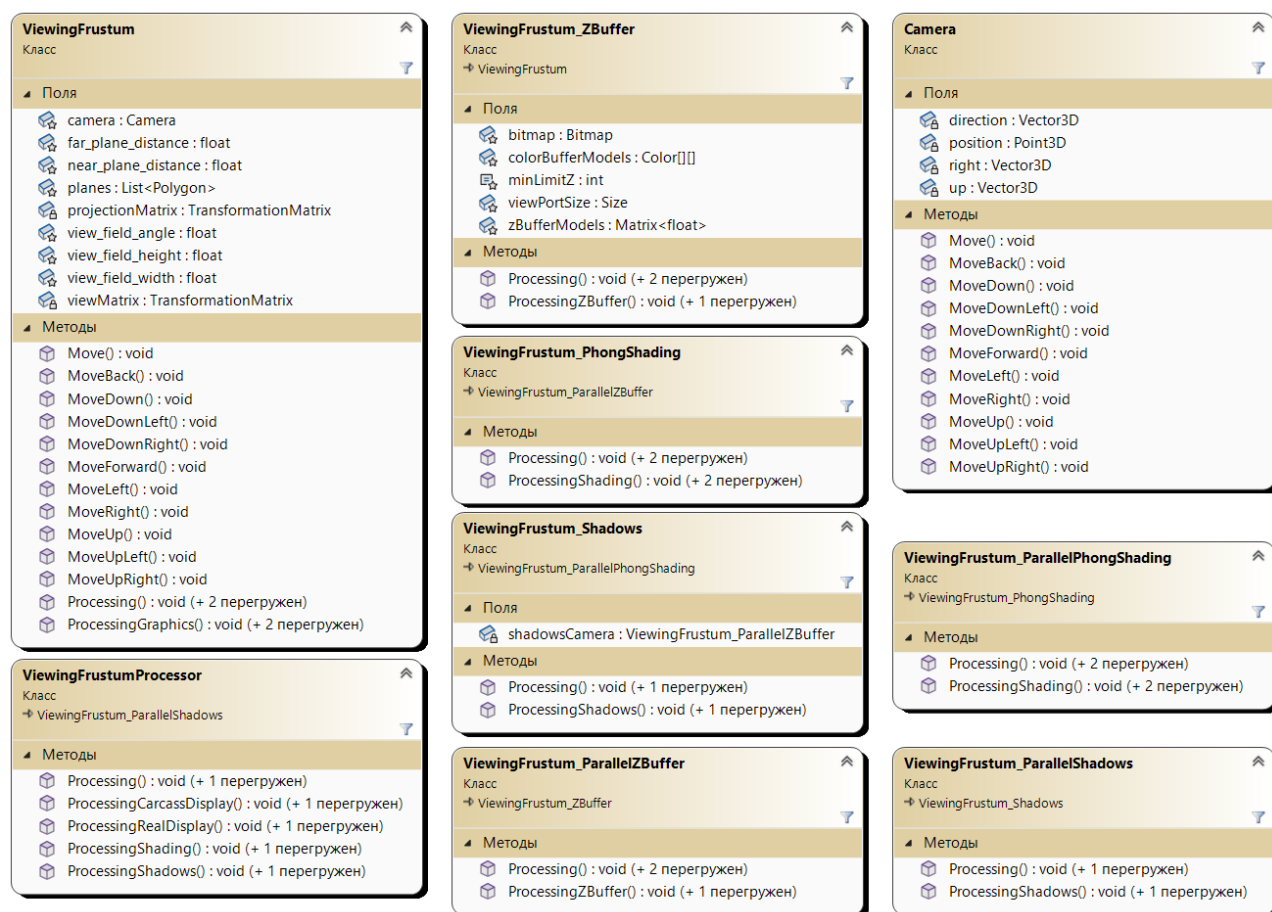


Рисунок 3.8 — Классы представления камеры

#### 9) классы аффинных преобразований:

- *Transformation* – базовый класс аффинного преобразования;
- *Move* – перемещение;
- *Rotate* – поворот;
- *Scale* – масштабирование.

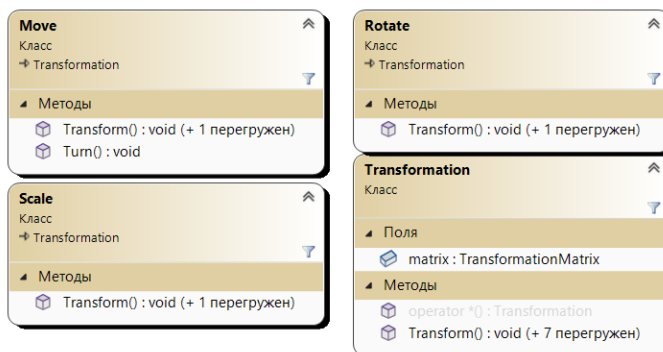


Рисунок 3.9 — Классы аффинных преобразований

#### 10) математические классы:

- *Matrix* – матрица;

- *TransformationMatrix* – матрица аффинного преобразования;
- *Vector3D* – трехмерный вектор.

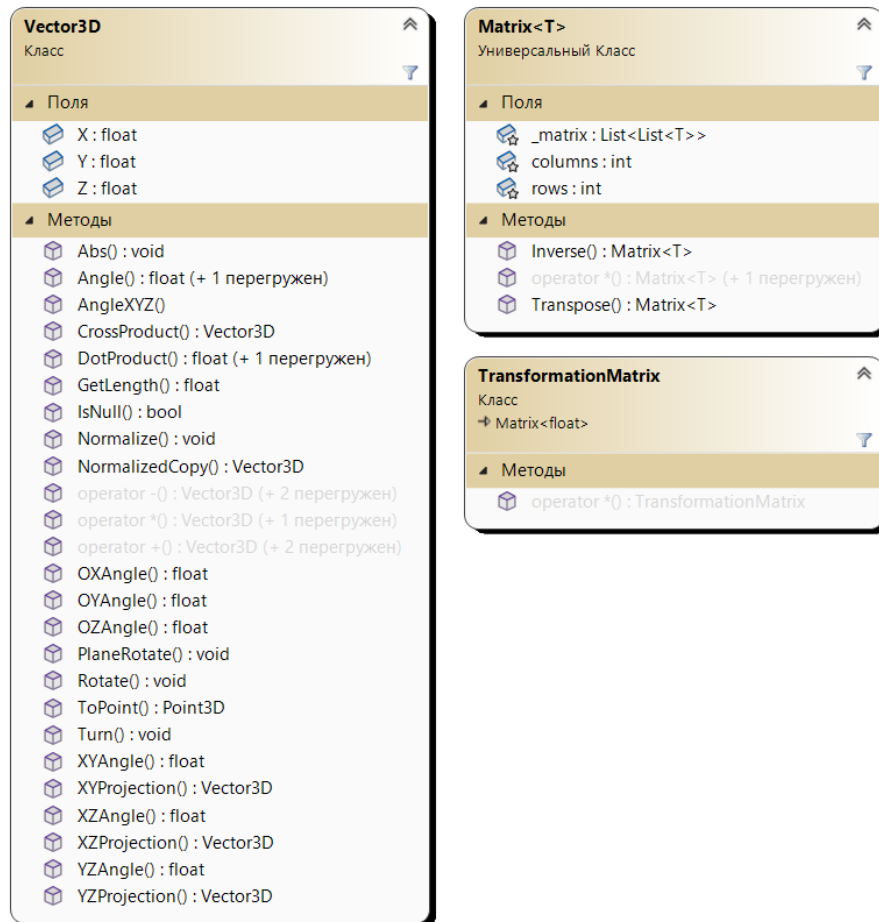


Рисунок 3.10 — Математические классы

### 3.3 Исходный код

Исходный код реализованной программы доступен в открытом сетевом репозитории.

### 3.4 Интерфейс программы

При запуске программы пользователю демонстрируется пустая сцена и панель, содержащая вкладки «Главная» и «Вид». Вкладка «Главная» позволяет пользователю добавлять примитивы на сцену, редактировать добавленные примитивы, очищать сцену и изменять режим отображения сцены. Вкладка «Вид» позволяет пользователю перемещать камеру, изменять спектральные характеристики и положение источника света.

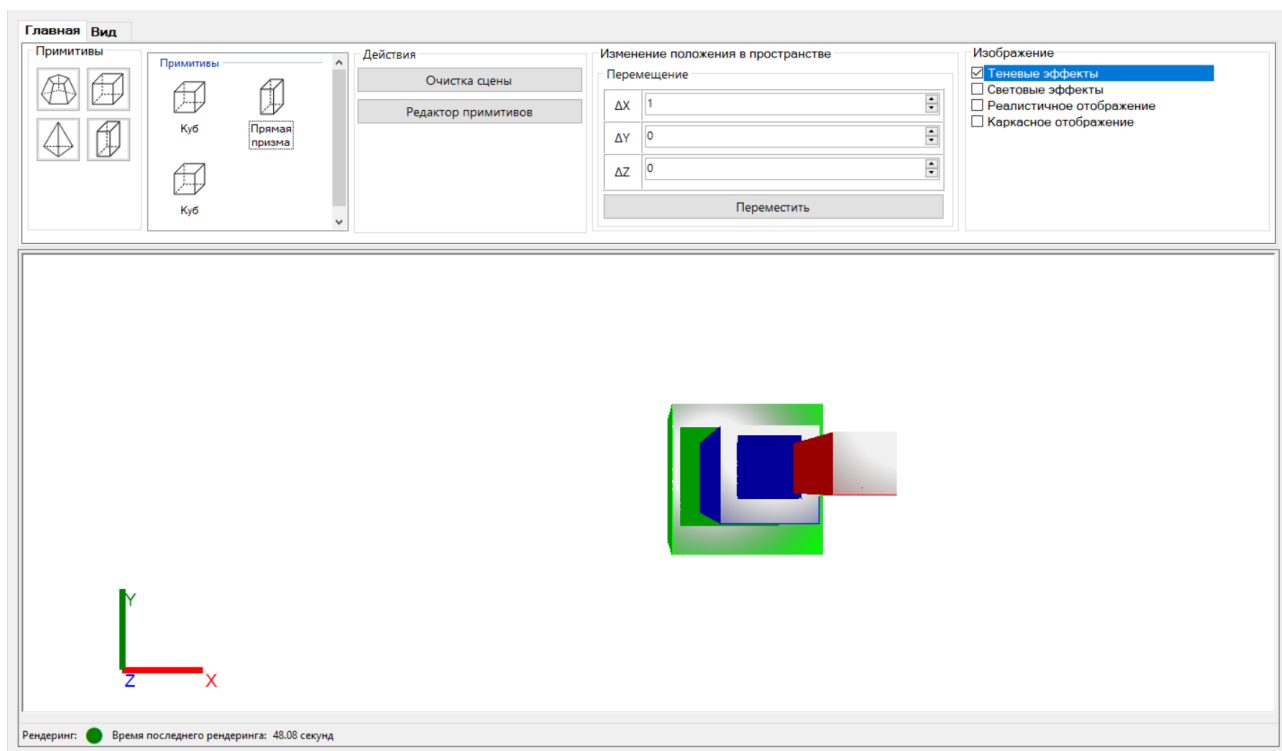


Рисунок 3.11 — Интерфейс главного окна программы во вкладке «Главная»

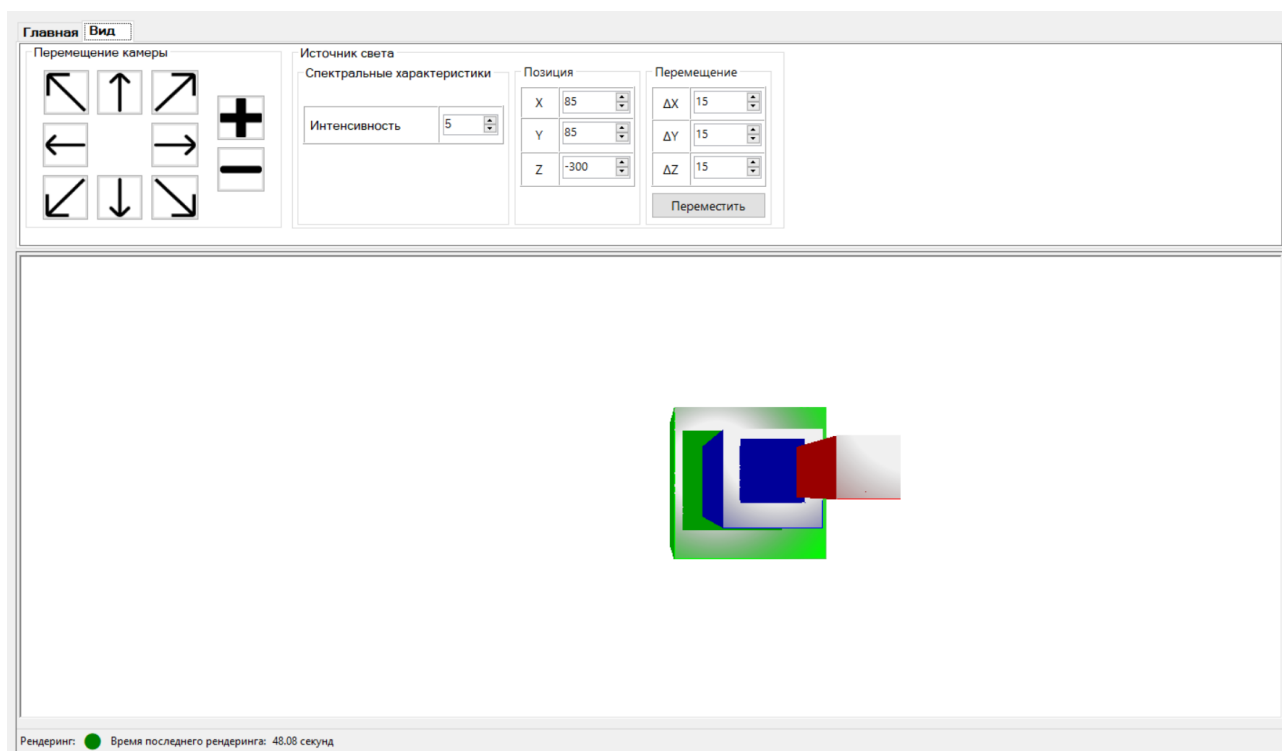


Рисунок 3.12 — Интерфейс главного окна программы во вкладке «Вид»

1) вкладка «Главная» содержит:

- группу «Примитивы», содержащую 4 кнопки, отвечающие за добавление на сцену многогранника, куба, прямой призмы и треугольной пирамиды соответственно;

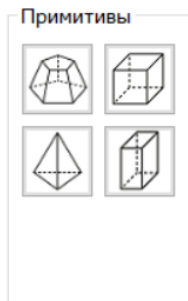


Рисунок 3.13 — Группа «Примитивы»

- коллекцию «Примитивы», отображающую присутствующие примитивы на сцене и позволяющую выбирать изменяемый примитив;

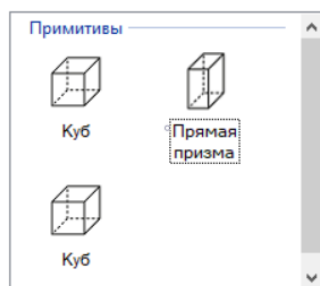


Рисунок 3.14 — Коллекция «Примитивы»

- группу «Действия», содержащую две кнопки: «Очистить», очищающую сцену от добавленных примитивов, «Редактор примитивов», открывающую окно редактирования примитивов на сцене;

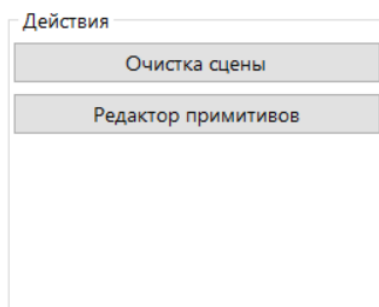


Рисунок 3.15 — Группа «Действия»

– группу «Изменение положения в пространстве», содержащую группу «Перемещение», позволяющую задавать параметры перемещения примитива и перемещать его нажатием кнопки «Переместить»;

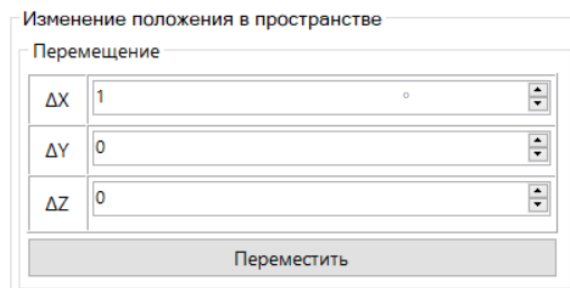


Рисунок 3.16 — Группа «Изменение положения в пространстве»

– группу «Изображение», содержащую список с возможностью выбора текущего режима отображения сцены между «Теневыми эффектами», «Световыми эффектами», «Реалистичным отображением» и «Каркасным отображением».

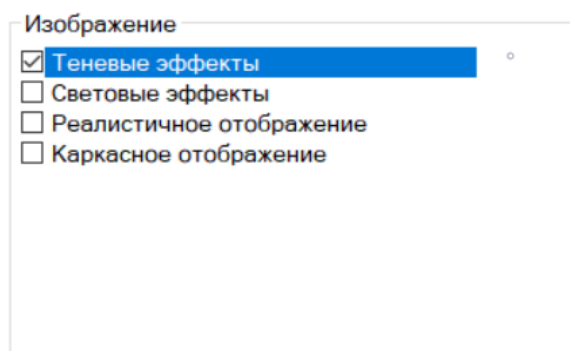


Рисунок 3.17 — Группа «Изображение»

2) вкладка «Вид» содержит:

– группу «Перемещение камеры», содержащую 10 кнопок и позволяющую переместить камеру нажатием на соответствующую кнопку;

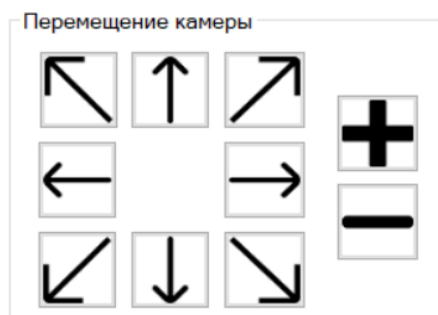


Рисунок 3.18 — Группа «Перемещение камеры»

– группу «Источник света», которая содержит группы «Спектральные характеристики», «Позиция» и «Перемещение», позволяющие задавать интенсивность потока света, испускаемого источником, изменять его позицию на сцене, определять параметры перемещения и перемещать нажатием кнопки «Переместить» соответственно.

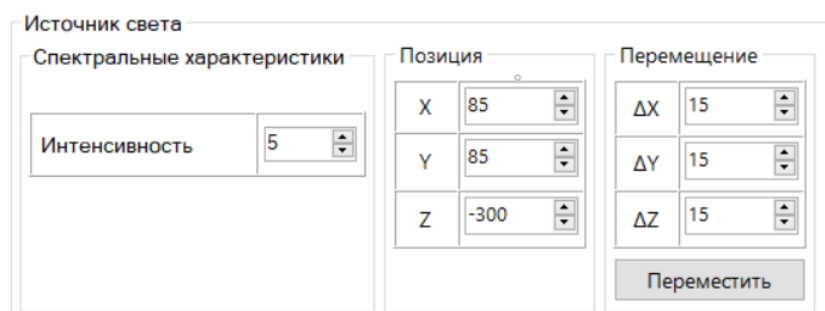


Рисунок 3.19 — Группа «Источник света»

При нажатии на кнопку «Редактор примитивов», расположенной в группе «Действия» вкладки «Главная» (рис. 3.15) будет открыто окно редактирования примитивов, в котором можно изменять его геометрические, спектральные, цветовые и информационные параметры. Для каждого из примитивов определен свой набор изменяемых геометрических параметров, потому некоторые параметры могут быть недоступны для изменения.



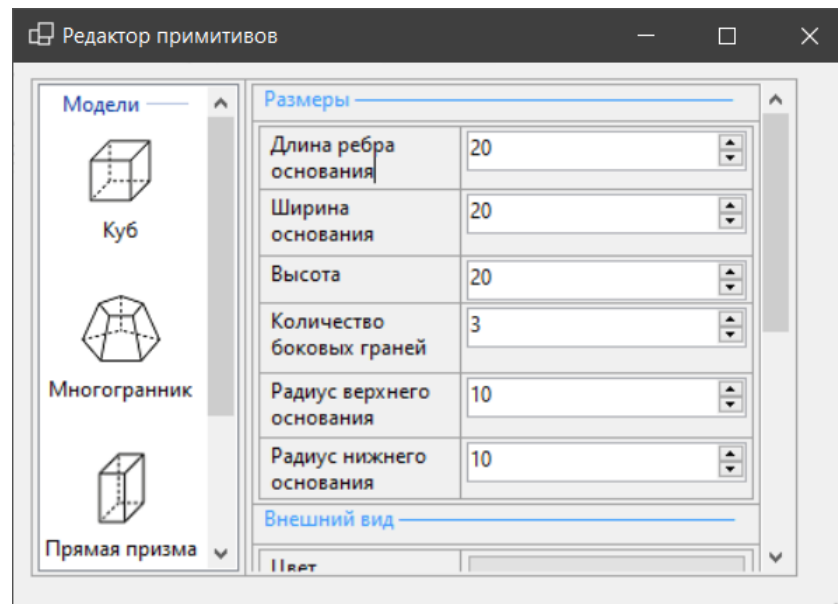


Рисунок 3.20 — Интерфейс окна редактирования примитивов (часть 1)

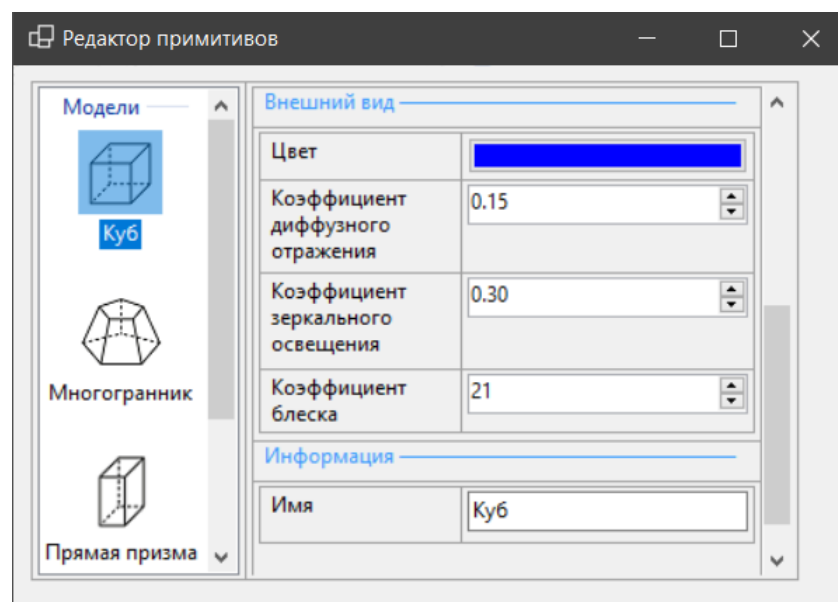


Рисунок 3.21 — Интерфейс окна редактирования примитивов (часть 2)

### 3.5 Вывод

В данной части были представлены выбор средств реализации и исходный код программы, описаны организация классов в программе и её интерфейс.

## **4 Исследовательская часть**

<цель исследования, на какой машине делали (указать ЦПУ, ОЗУ, ОС), желательно написать, как исследовали и при каких условиях; что получили в результате (таблицы + графики)>

В этой части приведены технические характеристики устройства, на котором проводилось исследование быстродействия программы, и результаты исследования.

### **4.1 Технические характеристики**

Спецификации устройства, использованного для тестирования:

- оперативная память 16 ГБ;
- процессор Intel(R) Core(TM) i7-9750H с тактовой частотой 2.60 ГГц;
- операционная система Windows 10 Домашняя 64-разрядная с версией 22H2.

Устройство было нагружено только встроенными приложениями и подключено в сеть электропитания.

### **4.2 Проведение исследования**

Целью исследования является определение зависимостей:

- времени визуализации сцены от количества полигонов;
- времени визуализации сцены от количества примитивов;
- времени визуализации сцены от размера примитива.

По результатам исследования должны быть составлены таблицы и построены графики зависимости.

#### **4.2.1 Зависимость времени визуализации от количества полигонов**

Исследование проводится с использованием многогранника, где каждая грань соответствует отдельному полигону. Для визуализации сцены выбран «теневой» режим, так как он является наиболее ресурсоемким из доступных в программе.

Для каждого многогранника время визуализации измеряется трижды, по-

сле чего вычисляется среднее значение. Все замеры производятся в секундах.

Результаты определения зависимости времени визуализации сцены от количества полигонов приведены в таблице 4.1.

Таблица 4.1 — Зависимость времени визуализации сцены от количества полигонов

Количество полигонов (шт.)	Время визуализации (сек.)
10	5,01
20	7,40
30	8,37
40	8,73
50	9,13
60	9,55
70	10,00
80	10,38
90	10,85
100	11,29

По таблице 4.1 был построен график 4.1, который наглядно демонстрирует результаты определения зависимости времени визуализации сцены от количества полигонов.

Исходя из графика можно сделать вывод, что зависимость близка к линейной.

### 4.3 Вывод

<что сделали и получили в результате, кратко>

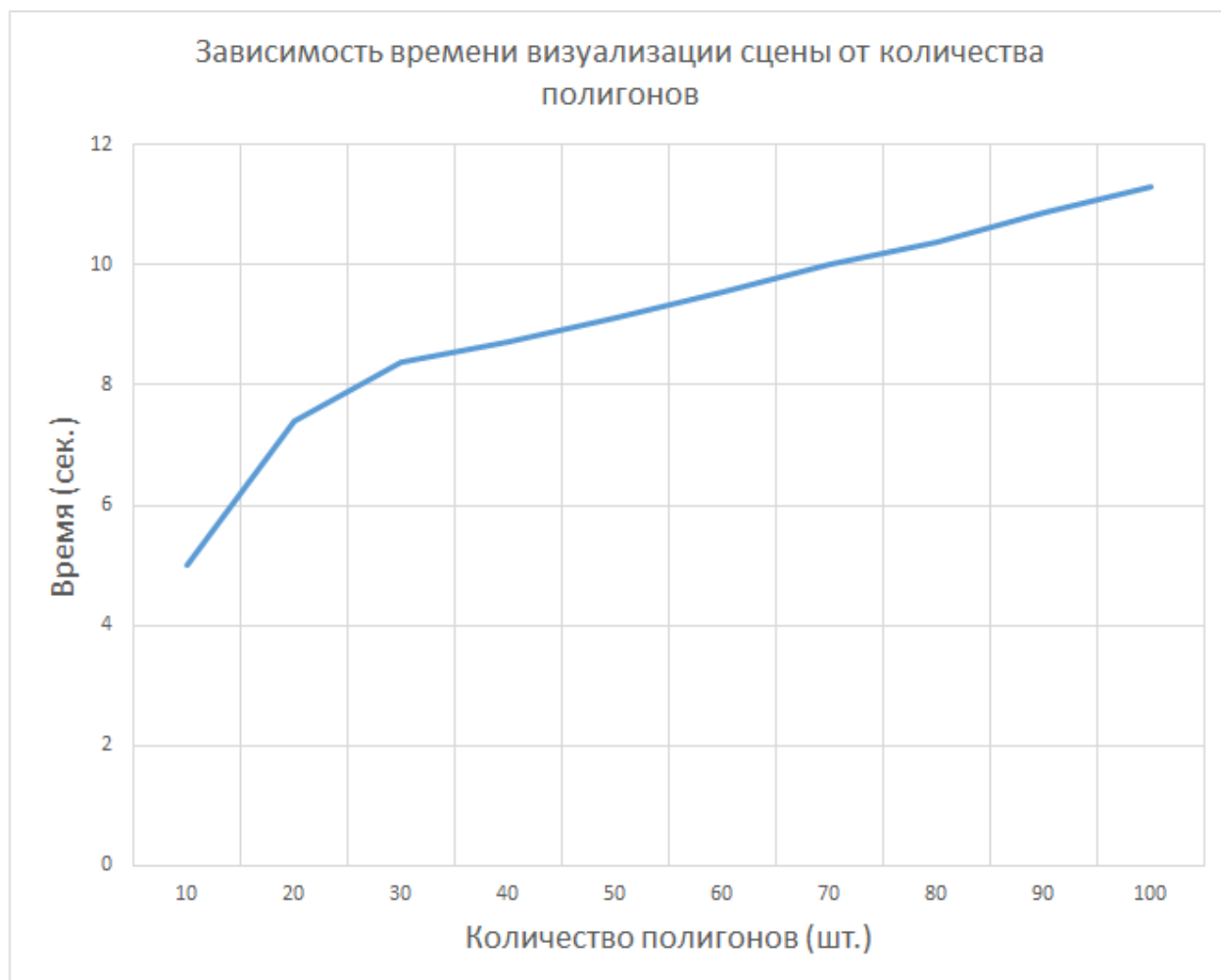


Рисунок 4.1 — Зависимость времени визуализации сцены от количества полигонов

# ЗАКЛЮЧЕНИЕ

<копируем цель и кратко описываем, что делали и получили в результате;  
указать, какие задачи выполнили>

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Литература 1.
2. Литература 2.
3. ...
4. Литература N.

## Приложение А

<тут всякие слишком большие ништяки, которые вы хотели бы добавить  
в отчет>