



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:*

*«Макетирование и визуализация загородной  
местности»*

Студент ИУ7-56Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

Вольняга М. Ю.  
(И. О. Фамилия)

Руководитель курсовой работы

\_\_\_\_\_  
(Подпись, дата)

Мальцева Д. Ю.  
(И. О. Фамилия)

*2023 г.*

# СОДЕРЖАНИЕ

<b>ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ</b>	<b>3</b>
<b>ВВЕДЕНИЕ</b>	<b>4</b>
<b>1 Аналитический раздел</b>	<b>5</b>
1.1 Способы определения моделей . . . . .	5
1.2 Методы представления трехмерных поверхностей . . . . .	6
1.3 Формализация объектов сцены . . . . .	8
1.4 Алгоритмы удаления скрытых линий и поверхностей . . . . .	9
1.5 Алгоритмы построения теней . . . . .	12
1.6 Постановка задачи . . . . .	14
<b>2 Конструкторский раздел</b>	<b>16</b>
2.1 Требования к программному обеспечению . . . . .	16
2.2 Разработка алгоритмов . . . . .	16
2.3 Выбор используемых типов и структур данных . . . . .	20
<b>3 Технологический раздел</b>	<b>21</b>
3.1 Средства реализации . . . . .	21
3.2 Сведения о модулях программы . . . . .	21
3.3 Реализация алгоритмов . . . . .	22
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>25</b>

## ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

Программное обеспечение — ПО

## ВВЕДЕНИЕ

ПО для макетирования и визуализации загородной местности может применяться в ландшафтном дизайне, архитектуре и планировании территории. Визуализация при планировании местности улучшает коммуникацию и снижает риск недопонимания между заказчиком и исполнителем [2].

Цель работы — разработка программного обеспечения для макетирования и визуализации загородной местности.

Для достижения поставленной цели требуется решить следующие задачи:

- формально описать структуру объектов;
- выбрать алгоритмы трехмерной графики для визуализации сцены;
- спроектировать ПО, позволяющее макетировать и визуализировать загородную местность;
- разработать программное обеспечение для макетирования и визуализации загородной местности;
- исследовать затраты реализации по времени выполнения.

# 1 Аналитический раздел

В данном разделе описаны: способы определения моделей, методы представления трехмерных поверхностей, формализация объектов сцены, алгоритмы удаления скрытых линий и поверхностей и алгоритмы построения теней.

## 1.1 Способы определения моделей

В системах трехмерного моделирования используются каркасные, поверхностные и объемные твердотельные модели. Правильный выбор метода определения моделей на сцене определяет размер и визуализацию модели на сцене.

**Каркасная модель** — это простейшая модель трехмерного объекта, представляющая собой совокупность вершин, соединенных между собой ребрами [1]. Главным недостатком данной модели является отсутствие информации о поверхности объекта, что делает невозможным разграничение внутренних и внешних граней, например, как на рисунке 1.1. Каркасная модель занимает меньше памяти и эффективна для простых задач [3].

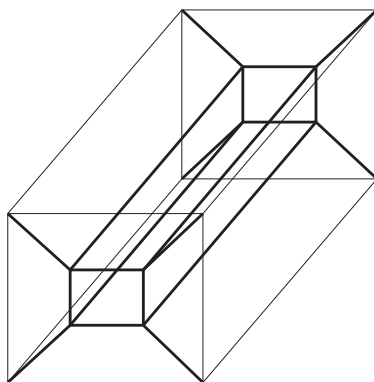


Рисунок 1.1 – Пример каркасной модели

**Поверхностная модель**, в отличие от каркасной, включает в себя не только вершины и ребра, но также поверхности, создавая визуальный контур объекта [1]. Каждый объект в данной модели обладает внутренней и внешней частью, как показано на рисунке 1.2.

В основу поверхностной модели положены два основных математических положения:

- любую поверхность можно аппроксимировать многогранником, где каж-

- дая грань представляет собой простейший плоский многоугольник [3];
- в модели допускаются не только плоские многоугольники, но и поверхности второго порядка, а также аналитически неопределяемые поверхности [3].

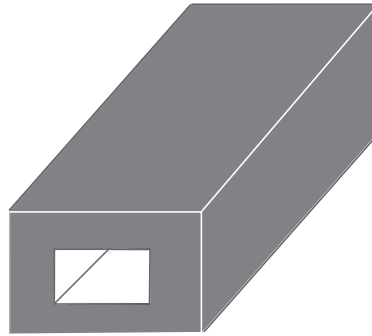


Рисунок 1.2 – Пример поверхностной модели

Недостаток поверхностной модели — отсутствует информация, о том, с какой стороны поверхности находится материал, а с какой пустота.

**Твердотельная модель** отличается от поверхностной тем, что включает информацию о расположении материала с обеих сторон поверхности [3].

### **Выбор определения моделей**

В данной задаче наиболее оптимальными являются поверхностные модели объектов, так как каркасные модели недостаточно полно представляют форму объекта, а твердотельные модели избыточны.

## **1.2 Методы представления трехмерных поверхностей**

**Метод полигональной сетки** представляет объект в виде связанной между собой сетки плоских многоугольников [1], как показано на рисунке 1.3.

**Метод параметрических бикубических кусков** использует математические формулы, описывающие координаты поверхностей. Бикубический — это термин, который указывает на то, что уравнения, описывающие координаты точек поверхности, содержат две пары параметров с показателями степени, не превышающими третьей [1]. Этот подход обеспечивает высокую точность при описании поверхности и требует меньше элементов для представления сложных форм, в сравнении с полигональными сетками. Однако алгоритмы, работающие с бикубическими кусками сложнее [1].

В рамках данной задачи выбран метод полигональной сетки. Этот выбор обоснован геометрической простотой объектов сцены и отсутствием необходимости использования сложных математических формул. Использование полигональной сетки позволит применять более простые алгоритмы для обработки объектов [1].

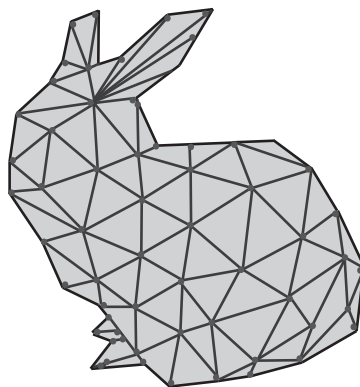


Рисунок 1.3 – Пример полигональной сетки, изображающей кролика

### **Способы описания полигональных сеток**

Наиболее распространенные методы представления полигональных сеток, рассматриваются в [4].

- 1) Список граней — распространенный метод представления трехмерных моделей, описывает объект как множество граней и вершин, где каждая грань имеет минимум 3 вершины. *Преимущества:* простота поиска вершин грани, динамическое обновление формы без изменения связности граней. *Недостатки:* трудности при операциях разрыва и объединения граней, а также проблемы с поиском граней.
- 2) Вершинное представление — это метод представления модели через множества вершин, которые связаны между собой. В качестве *преимущества* можно выделить его простоту. К *недостаткам:* отсутствие явной информации о гранях и ребрах, а также редкое использование в современных системах визуализации.
- 3) «Крылатое» представление — метод, представляющий модель как упорядоченное множество граней вокруг ребра. *Преимущество:* решение проблемы перехода от ребра к ребру через упорядоченное множество граней. *Недостаток:* высокие требования к памяти из-за увеличивающейся сложности структуры.

## Выбор способа описания полигональной сетки

Для работы выбран метод представления моделей через список граней, обеспечивающий ясное описание и удобный доступ к элементам сетки. Этот подход упрощает модификацию моделей, включая добавление, удаление и изменение граней и вершин.

### 1.3 Формализация объектов сцены

**Платформа** — это прямоугольный параллелепипед, состоящий из сетки квадратов, на которой размещаются объекты. Пользователь определяет границы сцены, указывая количество квадратов по ширине и длине. Размер каждого квадрата является постоянным и устанавливается программно. Цвет платформы — светло-зеленый.

#### Объекты, которые размещаются на платформе.

- 1) Программа включает два типа домов: «стандартный» (одноэтажное здание,  $3 \times 2$  квадрата) и «премиальный» (двухэтажное сооружение с гаражом,  $3 \times 3$  квадрата, включая гаражные стены, крышу и ворота). Каждый дом представляет собой единый объект, но состоит из нескольких компонентов: *крыша* (состоит из двух треугольников и двух трапеций, коричневого цвета), *стены* (формируются вертикальными плоскостями, бежевого цвета), *окна* (вертикальные плоскости, расположены в середине стены, голубого цвета), *дверь* (вертикальная плоскость, расположена посередине стены, коричневого цвета), *ворота* (вертикальная плоскость, расположена посередине двух стен, коричневого цвета).
- 2) Дороги занимают один квадрат платформы и имеют темно-серый цвет.
- 3) Тротуары аналогичны дорогам, светло-серого цвета.
- 4) Машины представляются как цельный объект из четырех колес, шести стекол и кузова. В программе имеется выбор модели машины, различного цвета. Компоненты машины: *кузов* (формируется набором плоскостей, имитирующих реальные автомобили, цвет красный или серый), *колеса* (представляют собой цилиндрические объекты, черного цвета), *окна* (плоскости, встроенные в кузов автомобиля, голубого цвета).



- 5) Деревья состоят из листвы и ствола, представляют цельный объект сцены и занимают один квадрат платформы. Листва имеет зеленый цвет, а ствол коричневый;
- 6) Кусты схожи с деревьями, но без ствола. Занимают один квадрат платформы.

Объекты, которые размещаются на платформе определены заранее и не могут быть изменены. ПО позволяет перемещать или удалять объекты на сцене. Есть ограничение на размещение моделей на платформе, а именно, машина может размещаться только на дороге. Дома, дороги, тротуары, деревья и кусты могут размещаться только на свободных квадратах платформы. Размер платформы ограничен.

### **Источник света**

Для моделирования освещения в компьютерной графике обычно используются три основных типа источников света: точечные, направленные и общий свет (от англ. Ambient Light) [5]. В данной работе выбор сделан в пользу точечного источника света.

Точечный источник света излучает свет равномерно во все стороны из определенной точки в трехмерном пространстве, что позволяет эффективно управлять освещением и тенями на сцене, учитывая положение объектов относительно источника света. Положение источника света устанавливается относительно текущей точки наблюдения с помощью последовательных поворотов по осям  $X$  и  $Y$  [5].

## **1.4 Алгоритмы удаления скрытых линий и поверхностей**

Алгоритмы удаления невидимых частей сцены классифицируются на основании следующих критериев:

- 1) выбор удаляемых частей: линий, ребер, поверхностей, объемов;
- 2) порядок обработки элементов сцены: удаление может быть определено процессом визуализации или выполняться в произвольном порядке;
- 3) зависимость от системы координат: существуют алгоритмы, работающие в пространстве объектов, где каждая из  $N$  граней объекта сравнивается

с остальными  $N - 1$  гранями (объем вычислений растет как  $N^2$ ), а также алгоритмы, работающие в пространстве изображения, когда для каждого пикселя изображения определяется, какая из  $N$  граней объекта видна (при разрешении экрана  $M \times M$  объем вычислений растет как  $M^2 \times N$ ) [6].

### **Алгоритм Робертса**

Алгоритм применяется к выпуклым объектам в трехмерном пространстве, представленным как многогранники, образованные пересечением плоскостей [6].

Алгоритм Робертса состоит из следующих шагов [6].

- 1) Удаление нелицевых плоскостей перед определением видимости.
- 2) Сравнение каждого ребра с каждым объектом (вызывает значительный объем вычислений, растущий квадратично по числу объектов в сцене).
- 3) Вычисляются новые ребра от пересечения объектов.

### **Z-буферный алгоритм удаления поверхностей**

Алгоритм представляет собой расширение обычного буфера кадра, который хранит цвета каждого пикселя. В Z-буферном алгоритме, дополнительно к каждому пикселю присваивается значение глубины (Z-координата). При добавлении нового пикселя в буфер, сравнивается его Z-координата с той, что уже находится в буфере. Если новый пиксель ближе к наблюдателю (т.е. его Z-координата больше, рисунок 1.4), его атрибуты и Z-координата записываются в буфер [6]. Пример работы алгоритма на рисунке 1.5.

Преимущества Z-буферного алгоритма: поддерживает сцены произвольной сложности, отсутствует необходимость в сортировке, что требуется в других алгоритмах, вычислительная сложность линейно зависит от числа анализируемых поверхностей. Недостаток Z-буферного алгоритма: требует значительного объема памяти для буферов [6].

### **Алгоритм Варнока**

Алгоритм Варнока использует принцип когерентности, по которому большие области изображения однородны. Алгоритм работает в пространстве изображения. Основная идея — разбивать окно на подокна до тех пор, пока

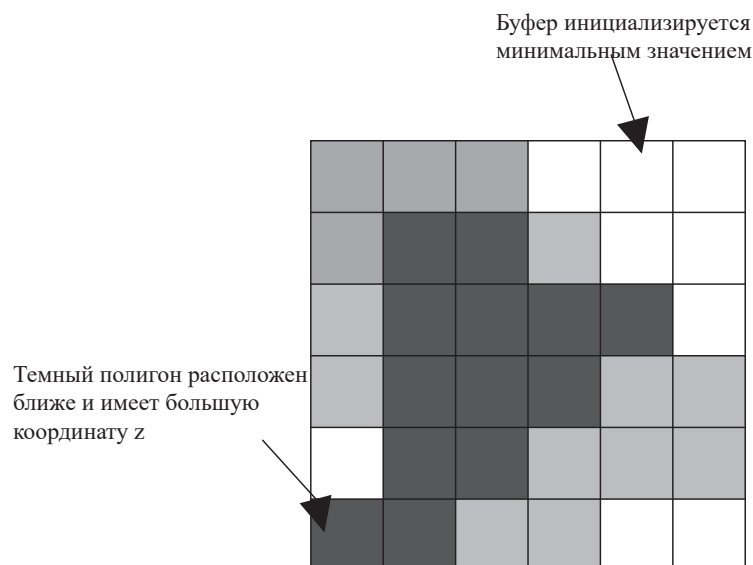


Рисунок 1.4 – Визуализация состояния Z-буфера

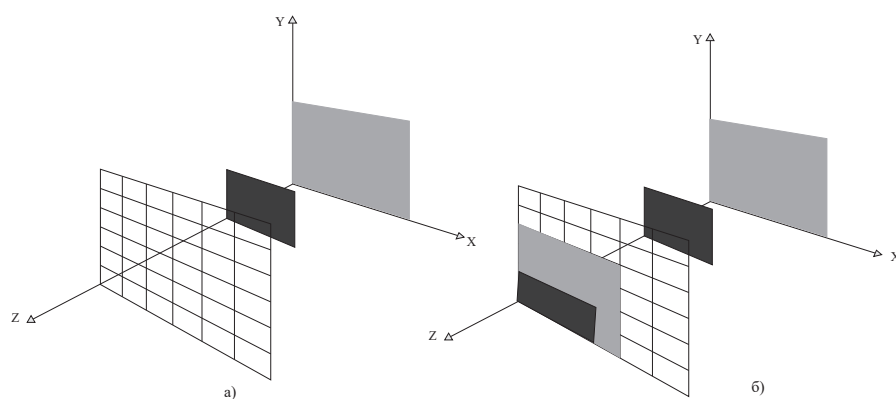


Рисунок 1.5 – Работа алгоритма с Z-буфером

их содержимое не станет достаточно простым для визуализации, или размер подокна не достигнет заданного предела разрешения [6; 7].

Конкретная реализация зависит от метода разбиения окна и критерия, определяющего простоту содержимого окна. В оригинальной версии алгоритма окно делится на четыре равных по размеру подокна. Вариант Вейлера и Азертон предполагает разбиение окна по ребрам изображаемых многоугольников. Эффективность данного алгоритма зависит от сложности сцены. Использование полигональной сетки может замедлить его выполнение [7].

### **Алгоритм прямой и обратной трассировки лучей**

Алгоритмы используются для отслеживания траектории лучей от источника света до камеры, учитывая взаимодействие с объектами на пути.

Прямая трассировка лучей строит траектории лучей от всех источников

света до всех точек сцены, включая те, которые не попадают в камеру. Из-за этого, метод считается неэффективным [8].

Обратная трассировка лучей — алгоритм при котором отслеживание лучей осуществляется не от источников света, а в обратном направлении от точки наблюдения. Учтены только лучи, влияющие на формирование изображения (см. рисунок 1.6). Недостатки алгоритма включают неучет вторичного освещения, высокую вычислительную стоимость и дискретность первичных лучей. Тем не менее, алгоритм поддерживает расчет теней, многократных отражений и преломлений [8].

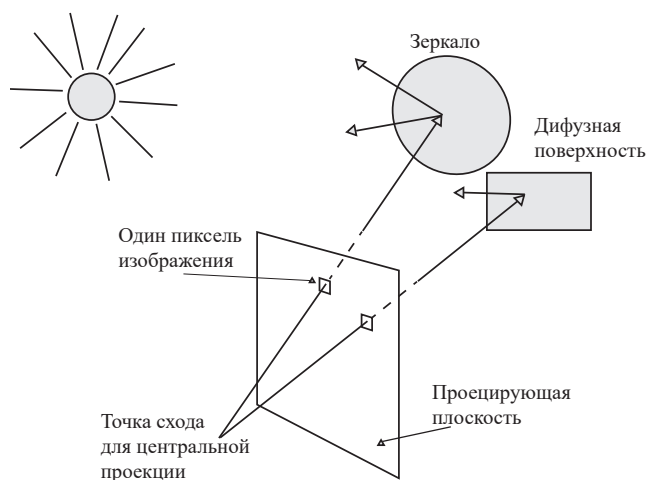


Рисунок 1.6 – Схема обратной трассировки лучей

## Выбор алгоритма удаления скрытых линий и поверхностей

На основании рассмотренных алгоритмов, Z-буфер является оптимальным выбором для удаления невидимых линий и поверхностей. Он обеспечит быструю обработку сцен любой сложности и адаптируется под изменения в освещении и закраске. При небольшом размере изображения Z-буфер эффективно обрабатывает большое количество объектов без проблем с памятью.

## 1.5 Алгоритмы построения теней

### Модификация Z-буфера

В алгоритмах отображения сцены можно адаптировать Z-буфер для отслеживания теней. Изначально, этот буфер хранит данные о глубине каждого пикселя, но для точного отображения теней нужен дополнительный этап.

Первым шагом камера устанавливается на место источника света, позволяя занести данные о глубине каждой точки в теневой буфер. Это гарантирует,

что все отмеченные точки расцениваются как освещенные [9].

Далее формируется изображение с точки зрения наблюдателя. На этом этапе каждый пиксель анализируется с учетом его расстояния до источника света и сопоставляется с данными из теневой карты. Если расстояние до пикселя больше значения из теневой карты, пиксель считается находящимся в тени и отображается с акцентом на рассеянном свете, это можно увидеть на примере рисунка 1.7, точка  $B$  имеет большее расстояние до источника света, чем восстановленный из карты теней, следовательно точка  $B$  будет в тени, а точка  $A$  находится на том же расстоянии от источника света, что и восстановлено из карты теней и не будет затемнена. В итоге, модификация Z-буфера позволяет эффективно учесть тени при рендеринге сцены, одновременно выполняя удаление невидимых поверхностей [9].

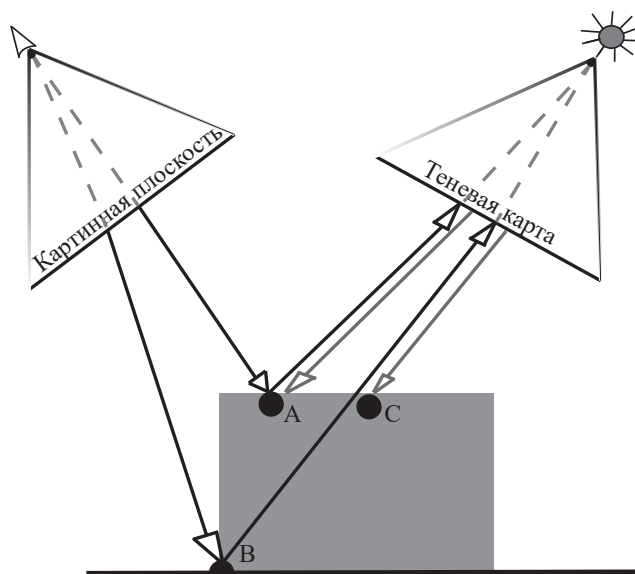


Рисунок 1.7 – Пример определения тени, алгоритмом модификации Z-буфера

### Модификация Вейлера-Азертона

На первом шаге с помощью алгоритма удаления невидимых поверхностей выбираются освещенные грани, т. е. грани, которые видны из положения источника света. Для увеличения эффективности, в памяти хранятся именно эти грани, что позволяет избежать удвоения обрабатываемых граней для выпуклого многоугольника. Освещенные многоугольники помечаются и преобразуются к исходной ориентации, где они приписываются к своим прототипам в качестве многоугольников детализации поверхности. Чтобы избежать появления ложных теней, сцену рассматривают в пределах видимого или отсекающего объема, определенного положением источника света [9].

На втором шаге объединенные данные о многоугольниках обрабатываются из положения наблюдателя. Если какая-то часть не освещена, применяется соответствующее правило затенения. Если источников света несколько, то используется несколько наборов освещенных граней [9].

### Выбор алгоритма построения теней

Выбор модифицированного Z-буфера для построения теней в данном контексте является оптимальным. Это обусловлено его эффективностью и совместимостью с уже используемым Z-буфером для удаления невидимых поверхностей. Использование одной и той же технологии упрощает интеграцию.

## 1.6 Постановка задачи

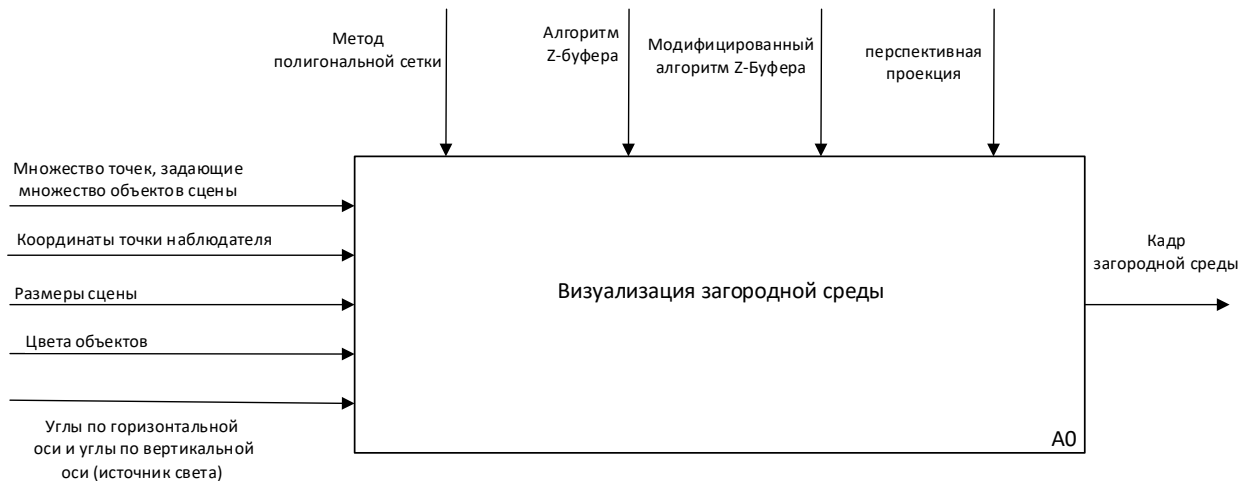


Рисунок 1.8 – Диграмма IDEF0 постановки задачи нулевого уровня

## Вывод

В представленном разделе были выбраны: поверхностные модели для описания трехмерных объектов, полигональные сети с использованием списка граней для представления трехмерных поверхностей, а также удаление невидимых ребер алгоритмом Z-буфера и построение теней с использованием модификации Z-буфера. Были формализованы объекты сцены.

Сравнение алгоритмов удаления невидимых линий и поверхностей приведено в таблице 1.1, в строке «Вычислительная сложность»  $N$  и  $C$  означают, количество граней и количество пикселей, соответственно.

Таблица 1.1 – Сравнение алгоритмов удаления невидимых линий и поверхностей.

	<b>z-буфер</b>	<b>Обратная трассировка лучей</b>	<b>Варнок</b>	<b>Робертс</b>
<b>Вычисл. сложность</b>	$O(CN)$	$O(CN)$	$O(CN)$	$O(N^2)$
<b>Производит. при сложной сцене</b>	Высокая	Низкая	Средняя	Низкая
<b>Рабочее пространство</b>	Изображение	Изображение	Изображение	Объектное
<b>Использ. рекурсивных вызовов</b>	Нет	Да	Да	Нет

## 2 Конструкторский раздел

В этом разделе представлены требования к программному обеспечению, а также схемы алгоритмов, выбранных для решения поставленных задач и описаны структуры данных, которые будут использоваться при реализации программного обеспечения.

### 2.1 Требования к программному обеспечению

Программа должна обладать следующей функциональностью:

- создавать сцену с определенным размером;
- размещать, удалять и перемещать отдельные объекты сцены;
- добавлять источник света;
- поворачивать, перемещать и масштабировать сцену с объектами.

Программа должна корректного реагирования на все действия пользователя.

### 2.2 Разработка алгоритмов

Общий алгоритм построения изображения показан на рисунке 2.1.

На рисунке 2.2 представлена схема алгоритма поиска Z-буфера. На вход алгоритму подаются следующие параметры: буфер глубины, буфер кадра, массив с полигонами сцены и атрибуты этих полигонов.

На рисунке 2.2 представлена схема алгоритма поиска Z-буфера. На вход алгоритму подаются следующие параметры: буфер глубины, буфер кадра, массив с полигонами сцены и атрибуты этих полигонов.

На рисунке 2.4 показана диграмма IDEF0 построения изображения первого уровня.



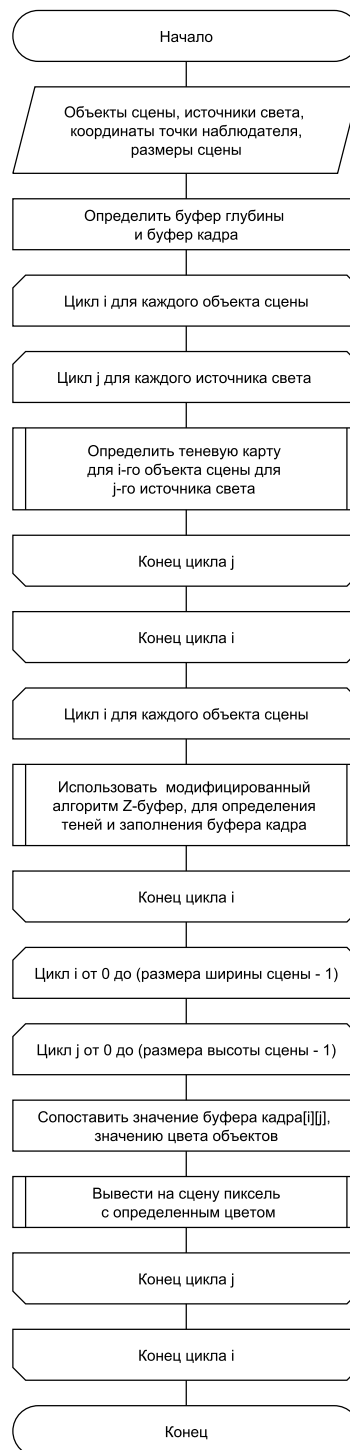


Рисунок 2.1 – Схема общего алгоритма построения изображения

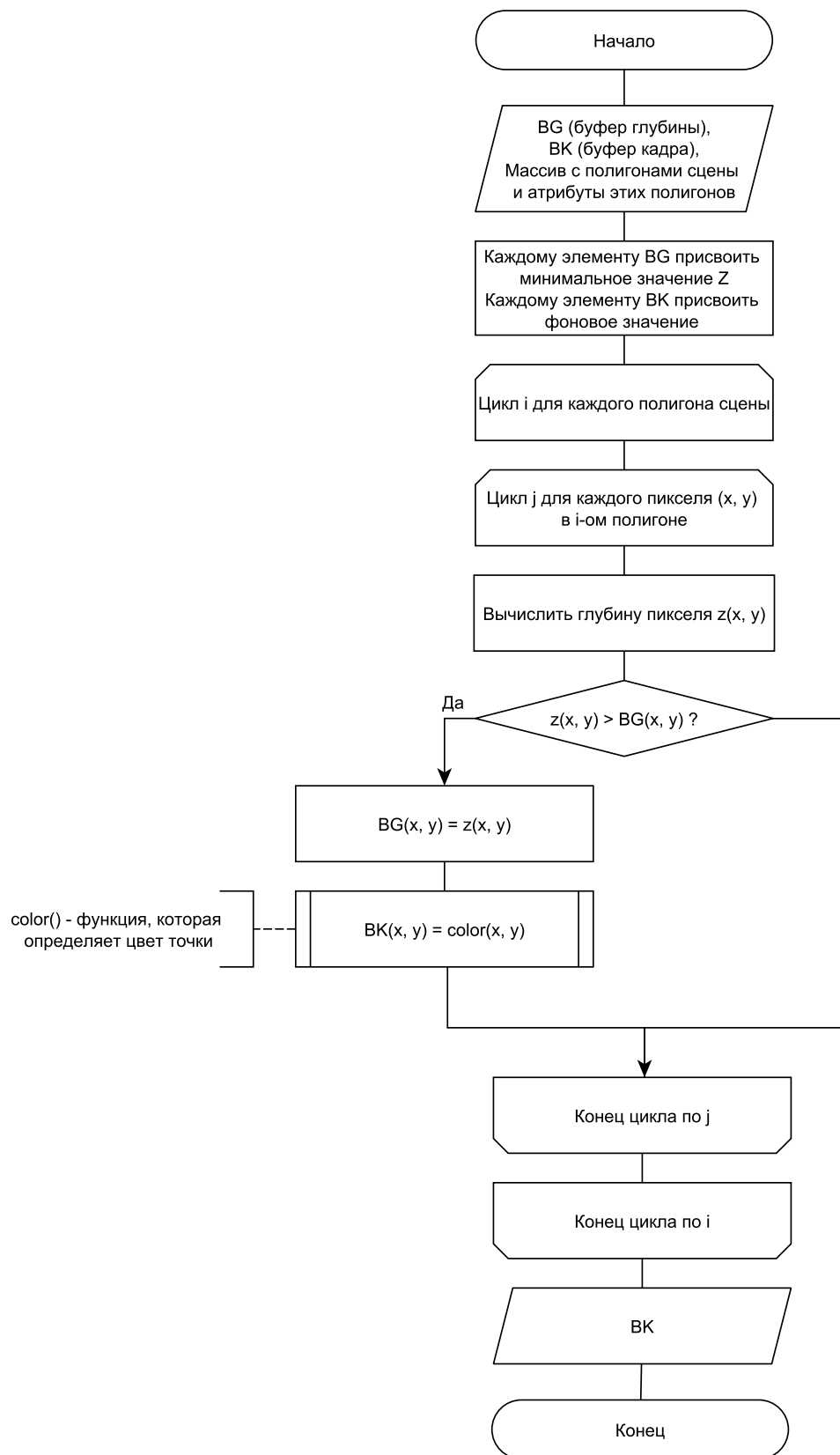


Рисунок 2.2 – Схема алгоритма Z-буфера

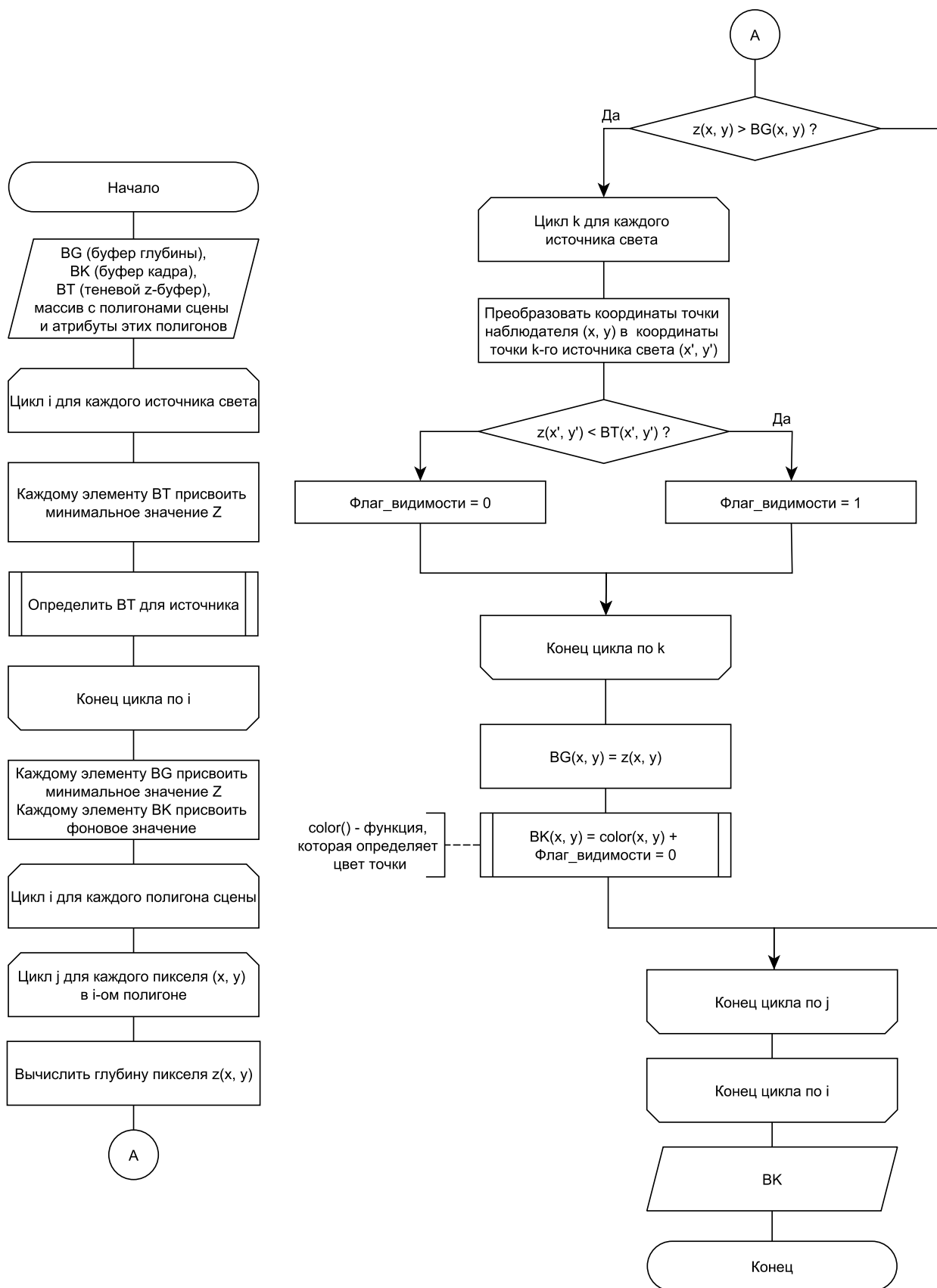


Рисунок 2.3 – Схема модифицированного алгоритма Z-буфера

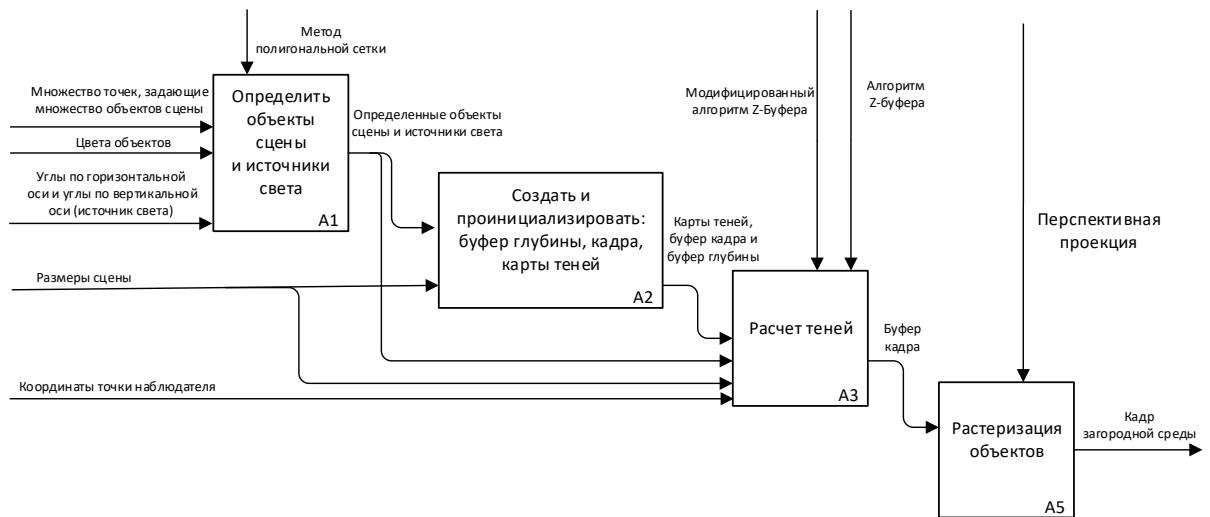


Рисунок 2.4 – Диграмма IDEF0 построения изображения первого уровня

## 2.3 Выбор используемых типов и структур данных

- 1) Точка задается координатами  $(x, y, z)$ .
- 2) Полигон задается тремя точками и храниться в массиве.
- 3) Объекты сцены зада полигонами в виде массива.
- 4) Источник света задается углом, по осям  $X$  и  $Y$  относительно точки наблюдателя.

## Вывод

Было спроектировано программное обеспечение, в ходе которого были представлены требования к программному обеспечению, а также схемы алгоритмов, выбранных для решения поставленных задач и описаны структуры данных, которые будут использоваться при реализации программного обеспечения.

## 3 Технологический раздел

В данном разделе будут приведены средства реализации, сведения о модулях программы, листинги кода реализации алгоритмов и интерфейс программного обеспечения.

### 3.1 Средства реализации

В качестве языка программирования для разработки программного обеспечения был выбран язык *C++* [10]. Данный выбор обусловлен следующим:

- 1) возможность реализовать все алгоритмы, выбранные в результате проектирования.
- 2) поддержка языком всех типов и структур данных, выбранных в результате проектирования;

### 3.2 Сведения о модулях программы

Программа состоит из трех программных модулей:

- 1) `main.cpp` — файл, который включает в себя точку входа программы.
- 2) `mainwindow.cpp`, `mainwindow.h` — модуль пользовательского интерфейса;
- 3) `mainwindow.ui` — форма пользовательского интерфейса;
- 4) `Drawer.cpp`, `Drawer.h` — модуль вывода результата работы программы на экран;
- 5) `Facade.cpp`, `Facade.h` — модуль реализующий действия пользователя;
- 6) `Platform.cpp`, `Platform.h` — модуль реализующий действия с сценой (платформой на которую устанавливаются объекты);
- 7) `Model.cpp`, `Model.h` — модуль реализующий действия с объектами;
- 8) `Point.cpp`, `Point.h` — модуль реализующий действия с точками объекта;

- 9) `Polygon.cpp`, `Polygon.h` — модуль реализующий действия с полигонами объекта;
- 10) `Vertex.cpp`, `Vertex.h` — модуль реализующий действия с вершинами объекта;
- 11) `Light.cpp`, `Light.h` — модуль реализующий действия с источником света;

### **3.3 Реализация алгоритмов**

В листинге 3.1 приведен фрагмент реализации модифицированной версии алгоритма Z-буфера для построения теней.

Листинг 3.1 – фрагмент реализации модифицированного алгоритма Z-буфера

```

1
2 void Drawer::zBufForModel(vector<Polygon> &polygons,
   vector<Vertex> &vertices,
3                               Eigen::Matrix4f &trans_mtr, size_t
   color, Platform *scene, size_t
4                               buf_w,
   size_t buf_h)
5 {
6     /* ... */
7     for (size_t cur_polygon = 0; cur_polygon < polygons.size();
   cur_polygon++)
8     {
9         Eigen::MatrixXf cord_vec(3, 4);
10        vector<size_t> cur_point =
   facets.at(cur_polygon).getUsedVertices();
11        for (int i = 0; i < 3; i++)
12        {
13            dotsArr[i] =
   vertices.at(cur_point.at(i)).getPosition();
14            cord_vec.row(i)
   << dotsArr[i].getXCoord(),
15            dotsArr[i].getYCoord(), dotsArr[i].getZCoord(),
16            1;
17        }
18        cord_vec *= dottrans_mtr;
19        /* ... */
20        for (int cur_y = (y1 < 0) ? 0 : y1;
   cur_y < ((y2 >= (int)buf_h) ? (int)buf_h - 1 : y2);
21            cur_y++)
22        {
23            double a_inc = 0;
24            if (y1 != y2)
25                a_inc = (double)(cur_y - y1) / (y2 - y1);
26            double b_inc = 0;
27            if (y1 != y3)
28                b_inc = (double)(cur_y - y1) / (y3 - y1);
29            int x_a = round(x1 + (x2 - x1) * a_inc);
30            int x_b = round(x1 + (x3 - x1) * b_inc);
31            double z_a = z1 + (z2 - z1) * a_inc;
32            double z_b = z1 + (z3 - z1) * b_inc;

```

```

33      /* ... */
34      for (int cur_x = x_a; cur_x <= x_b; cur_x++)
35      {
36          double curZ = z_a + (z_b - z_a) * (cur_x - x_a)
37              / (x_b - x_a);
38          if (curZ >= depthBuffer.at(cur_x).at(cur_y))
39          {
40              short visible = 0;
41              for (size_t i = 0; i < scene->getLightNum()
42                  && !visible; i++)
43              {
44                  /* ... */
45                  if (x < (int)shadowMap->size() && x >= 0
46                      &&
47                      y < (int)shadowMap->at(0).size() &&
48                      y >= 0 &&
49                      fabs(shadowMap->at(x).at(y) -
50                          newCoordinates(0, 2)) < 2)
51                      visible = 1;
52              }
53              // Обновляем буфер глубины текущим значением
54              z.
55              depthBuffer.at(cur_x).at(cur_y) = curZ;
56              if (scene->getLightNum())
57                  // Если пиксель видим, обновляем буфер
58                  кадра соответствующим цветом и
59                  признаком видимости.
60                  framebuffer.at(cur_x).at(cur_y) = color
61                      + visible;
62              else
63                  framebuffer.at(cur_x).at(cur_y) = color
64                      + 1;
65          }
66      }
67      // Аналогичные операции выполняются для следующей части
68      полигона (от y2 до y3)
69  }
70  }

```



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Лусяк В. В.* Основы геометрического моделирования. — 2022.
2. *Lovett A. Appleton K. W.-K. B.* Using 3D visualization methods in landscape planning: An evaluation of options and practical issues // Landscape and Urban Planning. — 2015. — С. 85—94.
3. *Донченко В. Ю. Ч. Е. Н.* Обзор и анализ методов построения геометрических моделей сложных конструкций. — 2014.
4. *Киселев А. В. В. Г. Н.* Способы представления и размещения трехмерных моделей для прототипирования ювелирных изделий // Материалы VI Международной научно-практической конференции (школы-семинара) молодых ученых. — 2020. — С. 840—842.
5. *Gambetta G.* Computer Graphics from Scratch. — 2021.
6. *Головнин А. А.* Базовые алгоритмы компьютерной графики. — 2016.
7. *Провкин В. А.* Построение реалистичных изображений при помощи алгоритма Варнока и сравнение эффективности различных его реализаций. — 2017.
8. *Янова Р. Д.* Метод прямой и обратной трассировки // Вестник науки и образования. — 2016. — С. 29—30.
9. *Куров А. В.* Конспект лекций по дисциплине компьютерная графика. — 2023.
10. Документация по C++20. — Режим доступа: <https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170/> (дата обращения: 20.11.2023).