|  |  |
| --- | --- |
| Gerb-BMSTU_01 | 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»

**ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Солопов Юрий Витальевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*фамилия, имя, отчество*

Группа \_\_\_\_ИУ7-46Б\_\_\_\_\_\_

Тип практики \_\_\_\_\_Технологическая\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Название предприятия \_\_\_\_\_\_МГТУ им. Н. Э. Баумана, каф. ИУ7\_\_\_\_\_\_\_\_\_\_\_\_

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_Солопов Ю. В.\_\_\_

*подпись, дата фамилия, и.о.*

Руководитель практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_Куров А.В.\_\_\_\_\_\_\_

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Москва, 2022 г.*

**Индивидуальное задание:**

Спроектировать программное обеспечение, представляющее из себя графический редактор трехмерных моделей, предоставляющий возможность редактирования моделей на уровне вершин, ребер и граней. Интерфейс программы должен позволять пользователю при помощи мыши выбрать на дисплее несколько вершин, ребер или граней (необходимо реализовать визуальное выделение выбранных элементов для наглядности) и затем произвести над ними любой вид трансформации. Помимо этого, реализовать операции переноса, поворота и масштабирования для всей модели в целом.

Предоставить возможность выбора примитивных моделей (куба, пирамиды) в качестве начальных, а также возможность добавления на сцену точечных источников освещения. Реализовать функционал сохранения/загрузки всей сцены в/из файл(а). Предоставить возможность отображения моделей как в каркасном, так и в полигональном режимах.

Оглавление

[Введение 5](#_Toc113359830)

[1. Аналитический раздел 6](#_Toc113359831)

[1.1. Формализация объектов синтезируемой сцены 6](#_Toc113359832)

[1.2. Выбор типа задания трехмерных моделей 6](#_Toc113359833)

[1.3. Выбор способа задания поверхностной модели 7](#_Toc113359834)

[1.4. Анализ способа выбора элементов модели на дисплее 9](#_Toc113359835)

[1.5. Анализ алгоритмов удаления невидимых линий и поверхностей 9](#_Toc113359836)

[1.5.1. Алгоритм Робертса 10](#_Toc113359837)

[1.5.2. Алгоритм Варнока 12](#_Toc113359838)

[1.5.3. Алгоритм, использующий Z-буфер 13](#_Toc113359839)

[1.5.4. Алгоритм обратной трассировки лучей 14](#_Toc113359840)

[1.5.5. Выбор оптимального алгоритма 15](#_Toc113359841)

[1.6. Анализ методов закраски граней 16](#_Toc113359842)

[1.6.1. Простая закраска 16](#_Toc113359843)

[1.6.2. Закраска по Гуро 16](#_Toc113359844)

[1.6.3. Закраска по Фонгу 17](#_Toc113359845)

[1.6.4. Выбор оптимального алгоритма закраски 17](#_Toc113359846)

[2. Конструкторский раздел 18](#_Toc113359847)

[2.1. Требования к программе 18](#_Toc113359848)

[2.2. Основные структуры данных 18](#_Toc113359849)

[2.3. Общий алгоритм визуального отображения сцены 19](#_Toc113359850)

[2.3.1. Аффинные преобразования 21](#_Toc113359851)

[2.3.2. Приведение к пространству камеры 22](#_Toc113359852)

[2.3.3. Проецирование вершины 23](#_Toc113359853)

[2.3.4. Отбрасывание нелицевых граней 23](#_Toc113359854)

[2.3.5. Преобразования трёхмерной сцены в пространство области изображения 24](#_Toc113359855)

[2.3.6. Алгоритм Z-буфера 24](#_Toc113359856)

[2.4. Простая модель освещения 26](#_Toc113359857)

[2.5. Алгоритм выбора составляющей части модели 27](#_Toc113359858)

[3. Технологический раздел 29](#_Toc113359859)

[3.1. Выбор средств реализации 29](#_Toc113359860)

[3.2 Структура и состав классов 29](#_Toc113359861)

[3.3 Сведения о модулях программы 32](#_Toc113359862)

[3.4 Интерфейс программы 32](#_Toc113359863)

[Заключение 34](#_Toc113359864)

[Список использованной литературы 35](#_Toc113359865)

# Введение

Компьютерная графика является одной из неотъемлемых частей современной жизни. Она используется повсеместно: в кинофильмах и мультфильмах, в компьютерных играх, а также для наглядного отображения данных.

В связи с этим возрастает потребность в инструментах, позволяющих создавать трехмерные модели различной сложности. На данный момент существует множество подобных инструментов, предоставляющих обширный спектр различных функций, упрощающих процесс создания 3d модели. Но такие редакторы, зачастую, имеют перегруженный интерфейс, требующий предварительного изучения.

Поэтому моей целью данного проекта является разработка редактора трехмерных моделей, имеющего интуитивно понятный интерфейс, и предоставляющего пользователю возможность производить базовые преобразования над полигональной моделью и ее составляющими частями.

В рамках реализации описанного проекта должны быть решены следующие задачи:

* Изучение и анализ существующих алгоритмов компьютерной графики удаления невидимых линий и поверхностей.
* Проектирование масштабируемой архитектуры программы и интуитивно понятного интерфейса.
* Проведение исследования на основе разработанной программы.

# Аналитический раздел

## Формализация объектов синтезируемой сцены

Сцена состоит из:

* **Источников света** – объекты сцены, которые представляют собой материальную точку, излучающую свет во всех направлениях.
* **Камеры** – специальный сцены, позволяющий просматривать содержимое сцены. Камера определяется двумя точками пространства: местоположением самой камеры, и точкой, к которой направлен “взгляд” камеры.
* **Моделей** – трехмерные объекты, расположенные на сцене.

## Выбор типа задания трехмерных моделей

Тип трехмерной модели определяет, как именно объект будет отображаться на сцене.

Модели могут задаваться в следующих формах:

* **Каркасная модель.** Это простейшая форма задания модели. Каркасная модель хранит информацию о вершинах и ребрах объекта. Основная проблема такого отображения объектов заключается в том, что модель не всегда однозначно передает представление о форме объекта, в следствии чего ее можно неправильно интерпретировать.
* **Поверхностная модель.** Этот тип модели хранит информацию о каждой поверхности объекта. Поверхность может описываться аналитически, либо задаваться другим способом. Одним из недостатков поверхностной модели является отсутствие информации о том, с какой стороны поверхности находится материал.
* **Твердотельная модель.** Данный тип задания модели отличается от поверхностного тем, что в твердотельных моделях к информации о поверхностях добавляется информация о том, с какой стороны расположен материал. Это можно сделать, например, путем указания направления внутренней нормали.

**Вывод:**

Использовать каркасный тип модели в моей задачи нецелесообразно, потому что такой тип в некоторых случаях может запутать пользователя, не давая полного представления об объекте.

Использовать твердотельную модель просто нет необходимости, т.к. пользователю моей программы не важно, с какой стороны от модели находится материал.

Поэтому я принял решение использовать полигональный тип модели, как самый подходящий для моей задачи. Тем не менее, моя программа будет предоставлять возможность отображения модели и в каркасной форме.

## Выбор способа задания поверхностной модели

Поверхностную модель можно задать несколькими способами:

**Аналитический способ** – для получения поверхности нужно вычислять функцию, зависящую от параметра.

**Полигональная сетка** – совокупностью вершин, ребер и граней, которые определяют форму объекта.

Рассмотрим известные способы хранения информации о сетке:

* **Вершинное представление.** Объект хранит множество вершин. Вершины указывают на другие вершины, с которыми они соединены.
* **Список граней.** Объект хранит множество граней и множество вершин. Каждая вершина при этом хранит информацию о соседних вершинах и о гранях, ее окружающих.
* ***Крылатое* представление.** Объект хранит информацию о вершинах, ребрах и гранях. При этом каждая вершина и грань хранят окружающие их ребра, а каждое ребро состоит из двух вершин (конечные точки), двух граней (по каждую сторону), и четырех ребер (*«крылья»* ребра).
* **Таблица углов.** Объект задается таблицей, хранящей вершины. Обход заданной таблицы неявно задает полигоны.

**Вывод:**

Использовать для моей задачи аналитический способ невозможно, т.к. задача предполагает возможность изменение модели посредством трансформации элементов модели.

Выбор конкретного способа хранения сетки напрямую зависит от поставленной задачи. Моя программа должна предоставлять возможность быстрого выбора и трансформации любых вершин, ребер и граней. Соответственно, решающим фактором является скорость выполнения этих операций. Поэтому полигональная сетка должна явно хранить информацию о каждой этой составляющей объекта.

Все вышеописанные способы, помимо крылатого представления, не хранят в явном виде информацию о ребрах, хотя в моей программе эта информация необходима. Крылатое представление же избыточно для моей задачи.

Поэтому я разработал собственный способ хранения информации о сетке. В качестве основы был взят список граней, к которому я добавил информацию о ребрах, которые определяются двумя конечными точками. Т.е. я решил использовать следующее представление модели:

* **Вершина**: Ее координаты в пространстве.
* **Ребро**: Две его конечные точки.
* **Грань**: Принадлежащие ей вершины и ребра.

Объект же определяется набором его граней, ребер и вершин.

## Анализ способа выбора элементов модели на дисплее

Специфика моей задачи предполагает, что программа должна предоставлять пользователю возможность выбора необходимых ему вершин, ребер и граней при помощи клика мышью по дисплею вблизи них. Более того, пользователь должен иметь возможность выбрать только те элементы фигуры, которые являются видимыми.

Для решения этой проблемы было принято решение помимо буфера кадра хранить еще и буфер граней, каждая ячейка которого будет указывать на грань, которая отображается в соответствующей ячейке буфера кадров. Буфер граней позволит за константное время определить, какую именно грань выбрал пользователь. И, при необходимости, можно будет быстро найти ближайшие к месту клика мыши точку или ребро.

## Анализ алгоритмов удаления невидимых линий и поверхностей

**Требования к алгоритму:**

При выборе алгоритма удаления невидимых линий необходимо в первую очередь учесть особенности поставленной задачи. Моя задача подразумевает, что алгоритм удаления невидимых линий будет быстродействующим, поскольку пользователь должен видеть плавную анимацию при трансформации моделей или камеры. Также немаловажное значение имеет сложность реализации определенного алгоритма. Кроме того, выбранный алгоритм должен позволять быстро и эффективно заполнять буфер граней, описанный в пункте 1.4.

При этом не имеет значения, в каком пространстве работает алгоритм, т.к. для моей задачи скорость важнее точности.

## Алгоритм Робертса

Данный алгоритм работает в объектном пространстве, решая задачу только с выпуклыми телами.

Алгоритм выполняется в 3 этапа.

1. **Этап подготовки исходных данных.** На данном этапе должна быть задана информация о телах. Для каждого тела сцены должна быть сформирована матрица тела V. Размерность матрицы – 4 \* n, где n – количество граней тела. Каждый столбец матрицы представляет собой четыре коэффициента уравнения плоскости , проходящей через очередную грань.

Матрица тела должна быть сформирована корректно, то есть любая точка, расположенная внутри тела, должна располагаться по положительную сторону от каждой грани тела. В случае, если для очередной грани условие не выполняется, соответствующий столбец матрицы надо умножить на −1. Для проведения проверки следует взять точку, расположенную внутри тела.

1. **Этап удаления ребер, экранируемых самим телом.** На данном этапе рассматривается вектор взгляда E = {0, 0,−1, 0}. Для определения невидимых граней достаточно умножить вектор E на матрицу тела V. Отрицательные компоненты полученного вектора будут соответствовать невидимым граням.
2. **Этап удаления невидимых ребер, экранируемых другими телами сцены.** На данном этапе для определения невидимых точек ребра требуется построить луч, соединяющий точку наблюдения с точкой на ребре. Точка будет невидимой, если луч на своем пути встречает в качестве преграды рассматриваемое тело. Если тело является преградой, то луч должен пройти через тело. Если луч проходит через тело, то он находится по положительную сторону от каждой грани тела.

Основным преимуществом данного алгоритма является точность вычислений. Она достигается за счет работы в объектном пространстве, в отличии от большинства других алгоритмов.

Серьезным недостатком является вычислительная трудоемкость алгоритма. В теории она растет как квадрат количества объектов. Поэтому при большом количестве объектов на сцене, этот алгоритм будет работать достаточно медленно. Но для решения этой проблемы можно использовать разные оптимизации для повышения эффективности, например, сортировку по z. Тем не менее, некоторые из оптимизаций крайне сложны, что затрудняет реализацию этого алгоритма.

Еще один весомый недостаток этого алгоритма - все тела сцены должны быть выпуклыми. Данная проблема также приводит к усложнению алгоритма, так как потребуется прибегнуть к проверке объектов на выпуклость и их разбиению на выпуклые многоугольники, что сильно замедлит алгоритм при большом количестве тел.

**Вывод:**

Алгоритм Робертса не подходит для решения поставленной задачи по следующим причинам:

* от программы не требуется той точности визуализации объектов, которую предоставляет алгоритм.
* на сцене может находиться множество объектов (зачастую - невыпуклых), что сильно замедлит скорость работы алгоритма. Таким образом, алгоритм не удовлетворяет требованиям к скорости выполнения алгоритма.
* реализация модификаций, позволяющих приблизить рост сложности алгоритма к линейной, очень труднозатратна.
* при использовании этого алгоритма возникают трудности с заполнением «буфера граней».

## Алгоритм Варнока

Основной идеей данного алгоритма является рекурсивное разбиении области экрана на более мелкие подобласти. Для каждой подобласти определяются связанные с ней многоугольники и те из них, видимость которых тривиальна, изображаются на экране. В случае невозможности однозначно определить видимость части многоугольника разбиение области повторяется, и для каждой из вновь полученных подобластей рекурсивно применяется процедура определения видимости. В результате работы алгоритма получаются области, содержащие не более одного многоугольника, либо разбиение продолжается до тех пор, пока размер области не станет равен одному пикселю. В этом случае для полученного пикселя вычисляется значение глубины каждого многоугольника (координата Z), и визуализируется тот из них, у которого значение этой координаты больше.

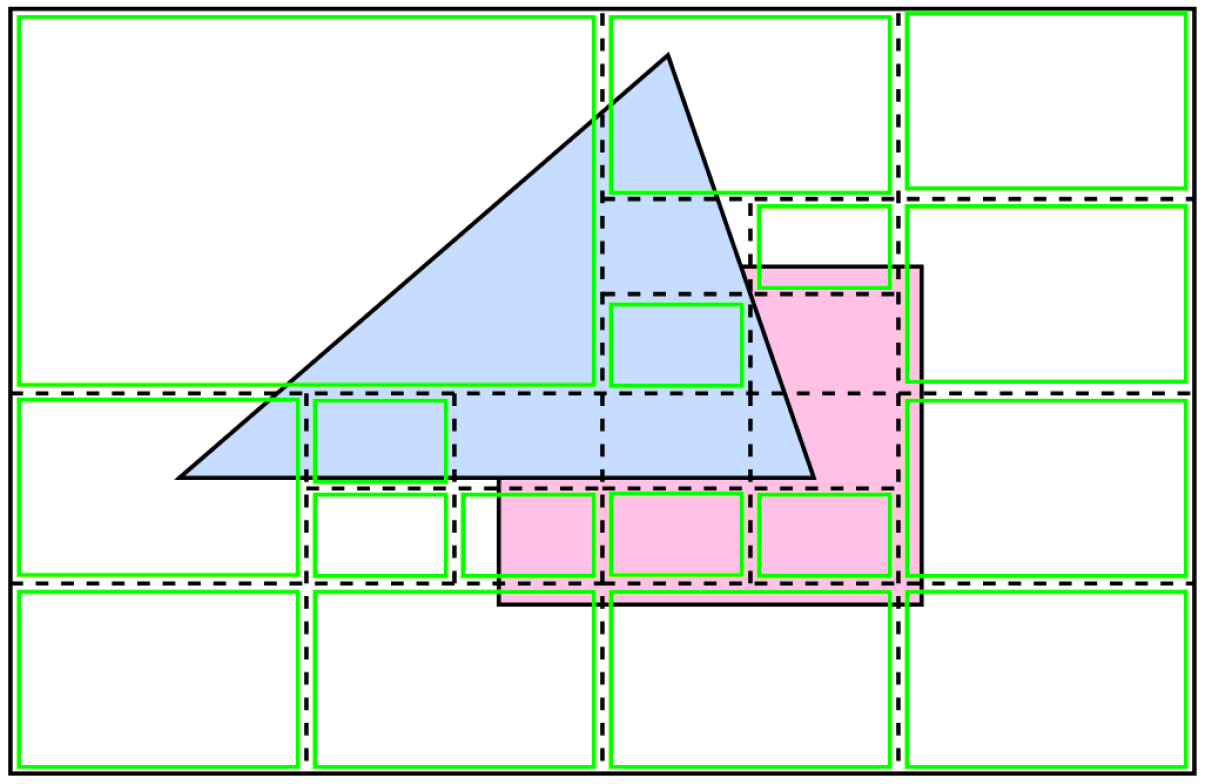


Рисунок 1. Алгоритм Варнока

Достоинством данного алгоритма является простота реализации и высокая эффективность в случае, если размеры перекрываемых областей невелики.

Его главным недостатком является использование рекурсивных вызовов, что значительно снижает скорость выполнения в случае больших размеров перекрываемых областей.

**Вывод:**

Алгоритм Варнока не подходит для решения моей задачи по следующим причинам:

* алгоритм зависит от количества перекрывающихся областей на сцене поэтому при его использовании отрисовка сцены может работать неоднородно.
* при использовании этого алгоритма возникают трудности с заполнением «буфера граней».

## Алгоритм, использующий Z-буфер

Данный алгоритм работает в пространстве изображения. Используется два буфера:

* буфер кадра, в котором хранятся атрибуты каждого пикселя в пространстве изображения;
* z-буфер, куда помещается информация о координате z для каждого пикселя.

В ходе работы алгоритма значение глубины каждого нового пикселя, заносимого в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в Z-буфер. Если это сравнение показывает, что новый пиксель расположен ближе к наблюдателю, чем пиксель, уже находящийся в буфере кадра, то новый пиксель заносится в буфер кадра и производится корректировка Z-буфера: в него заносится глубина нового пикселя. Если же значение глубины нового пикселя меньше, чем хранящееся в буфере, то осуществляется переход к следующей точке.

Несомненным плюсом данного алгоритма является его простота, которая не мешает решению задачи удаления поверхностей и визуализации их пересечения. Также в этом алгоритме отсутствие необходимости предварительной сортировки объектов по глубине, то есть они могут обрабатываться в произвольном порядке. Более того, время работы алгоритм линейно зависит от количества граней на сцене, что делает его одним из самых быстродействующих.

К недостаткам данного алгоритма относят необходимость выделения памяти под два буфера, каждый из которых имеет размер равный количеству пикселей на экране, но для современных компьютеров этот недостаток является незначительным.

**Вывод:**

Данный алгоритм наилучшим образом подходит для решения поставленной задачи, так как:

* он прост в реализации, что, помимо прочего, позволит грамотно и в полной мере его отладить.
* алгоритм способен быстро работать даже с множеством объектов на сцене.
* с помощью этого алгоритма можно очень просто реализовать заполнение «буфера граней», т.к. это можно производить вместе с заполнением двух других буферов.

## ****Алгоритм обратной трассировки лучей****

Идея данного алгоритма заключается в том, что для каждого пикселя картинной плоскости определяется ближайшая к нему грань. Для этого через пиксель выпускается луч, находятся все пересечения луча с гранями и среди пересечений выбирается ближайшее.

К достоинствам данного алгоритма можно отнести возможность получения изображения гладких объектов без аппроксимации их примитивами (например, треугольниками). Благодаря отслеживанию пути, пройденного лучом, появляется возможность реализовать глобальную модель освещения, учитывающую отражения и преломления света. Качество полученного изображения получается очень реалистичным, этот метод отлично подходит для создания фотореалистичных сцен. Также важным преимуществом этого алгоритма является возможность применения параллельных вычислений, т.к. расчет отдельной точки выполняется независимо от других точек.

Главным недостатком алгоритма трассировки является необходимость создавать большое число лучей, проходящих через сцену, которые могут раздваиваться на отраженный и преломленный лучи, для которых все вычисления повторяются. Это приводит к существенному снижению скорости работы программы.

**Вывод:**

Данный алгоритм не отвечает главному требованию – скорости работы. От реализуемого продукта не требуется высокой реалистичности синтезируемого изображения и возможности работы с поверхностями, заданными в математической форме. Также в моей программе не предполагается присутствие эффектов отражения и преломления света. Таким образом, при заметном замедлении работы программы, качество изображения заметно не улучшится. Указанные факты говорят о том, что использовать алгоритм обратной трассировки лучей для моей задачи нецелесообразно.

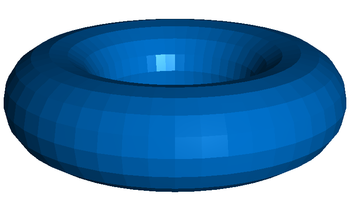
## Выбор оптимального алгоритма

С учетом изложенных выше заключений, в качестве алгоритма удаления невидимых линий и поверхностей был выбран алгоритм, использующий Z-буфер.

## Анализ методов закраски граней

Существуют три основных алгоритма, позволяющих закрасить полигональную модель: простая закраска, закраска по Гуро и закраска по Фонгу.

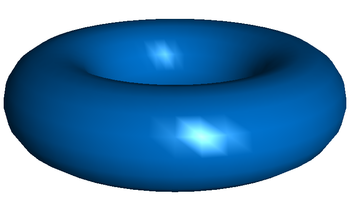
## Простая закраска

Суть данного алгоритма заключается в том, что для каждой грани объекта находится вектор нормали, и с его помощью в соответствии с выбранной моделью освещения вычисляется значение интенсивности, с которой закрашивается вся грань.

Данный метод закраски обладает большим быстродействием, однако все пиксели грани имеют одинаковую интенсивность, и сцена выглядит нереалистично. Тем не менее, этот метод крайне прост в реализации и совершенно не требователен к ресурсам.

Рисунок 2. Простая закраска

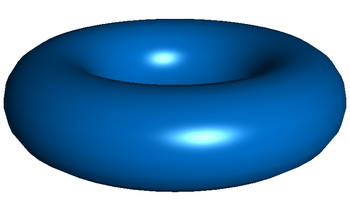
## Закраска по Гуро

Основа закраски по Гуро – билинейная интерполяция интенсивностей, за счет которой устраняется дискретность изменения интенсивности и создается иллюзия гладкой криволинейной поверхности. Иначе говоря, разные точки грани закрашиваются с разными значениями интенсивности.

С помощью этого метода получаются достаточно реалистичные изображения, однако все объекты кажутся матовыми.

Рисунок 3. Закраска по Гуро

## Закраска по Фонгу

Закраска Фонга по своей идее похожа на закраску Гуро, но ее отличие состоит в том, что в методе Гуро по всем точкам полигона интерполируются значения интенсивностей, а в методе Фонга – вектора нормалей, и с их помощью для каждой точки находится значение интенсивности.

Эта закраска требует больших вычислительных затрат, чем предыдущие, однако она позволяет достигнуть лучшей локальной аппроксимации кривизны поверхности и, следовательно, с ее помощью получается более реалистичное изображение.

Рисунок 4. Закраска по Фонгу

## Выбор оптимального алгоритма закраски

Моя задача предполагает, что пользователь должен четко видеть ребра и вершины редактируемой модели. А закраски по Фонгу и Гуро будут сглаживать ребра, тем самым мешая пользователю сосредоточиться на моделировании. К тому же, они будут замедлять время отрисовки сцены.

Таким образом, для моей задачи оптимальным выбором будет простой алгоритм закраски.

# Конструкторский раздел

В рамках данного раздела приведены основные требования к ПО, описаны основные используемые структуры данных и алгоритмы.

## Требования к программе

Разрабатываемое ПО должно предоставлять следующие возможности:

* Визуальное отображение сцены с объектами
* Добавление и удаление моделей
* Загрузка и сохранение сцены из\в файл(а)
* Выбор конкретной модели или её составляющей части\*
* Преобразование конкретной модели или её составляющей части\*

\* Составляющие части модели - вершины, ребра и грани.

## Основные структуры данных

Для формализации общего алгоритма синтеза изображения в данной программе, необходимо определить использующиеся в ней структуры данных. Ниже приведены основные структуры данных и их составляющие:

* Сцена:
  + Список моделей
  + Список источников света
  + Камеру
* Модель:
  + Название модели
  + Список ее вершин
  + Список ее ребер
  + Список ее граней
  + Вершину, определяющую ее центр
* Источник света:
  + Название источника света
  + Вершину, определяющую ее местоположение
  + Интенсивность света
* Камера:
  + Вершину, определяющую ее местоположение
  + Координаты точки, к которой направлен взгляд камеры
* Вершина:
  + Начальное ее местоположение
  + Матрица ее трансформации
* Ребро:
  + Вершины на его концах
* Грань:
  + Принадлежащие ей вершины
  + Принадлежащие ей ребра

## Общий алгоритм визуального отображения сцены

**Вход:** Сцена с объектами

**Выход:** Буфер кадра с построенной в нем сценой

Общая схема алгоритма изображена на рисунке 5:

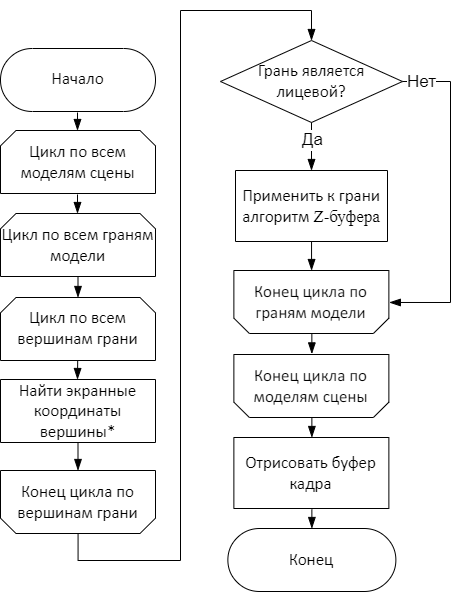


Рисунок 5. Алгоритм визуализации сцены

\* Поиск экранных координат вершины предполагает последовательное применение к начальным координатам вершины следующих преобразований:

* Аффинные преобразования
* Приведение вершины к пространству камеры
* Перспективные преобразования, при необходимости
* Преобразования в пространство области изображения

## Аффинные преобразования

Первым этапом получения экранных координат вершины является применение к ней аффинных преобразований. Это преобразование осуществляется путем умножения матрицы трансформации вершины на матрицу соответствующего аффинного преобразования.

В данном проекте над вершиной можно произвести следующие аффинные преобразования:

* **Перенос:**

Перенос в трехмерном пространстве задается значениями переноса вдоль осей OX, OY, OZ - dx, dy, dz соответственно. Матрица переноса имеет вид:

* **Поворот:**

Поворот описывается углом θ и осью вращения. Матрицы поворота имеют вид:

* + Вокруг оси OX:
  + Вокруг оси OY:
  + Вокруг оси OZ:
* **Масштабирование:**

Масштабирование в трехмерном пространстве задается значениями масштабирования вдоль осей OX, OY, OZ - kx, ky, kz соответственно. Матрица масштабирования имеет вид:

При этом важно заметить, что перед применением преобразований поворота и масштабирования необходимо предварительно осуществить перенос вершины так, чтобы центр соответствующего преобразования совпал с центром координат.

## Приведение к пространству камеры

Приведение к пространству камеры производится путем умножения матрицы трансформации вершины на матрицу преобразования камеры. При этом стоит учитывать, что над камерой возможно осуществления лишь 2 вида преобразования:

* **Перенос:**

Осуществляется аналогично переносу вершины, но все значения смещения предварительно умножаются на -1.

* **Поворот:**

Осуществляется аналогично повороту вершины, но все значения углов предварительно умножаются на -1.

## Проецирование вершины

После перехода в пространство камеры необходимо спроецировать вершину на картинную плоскость. В данном курсовом проекте используется перспективная и ортогональная проекции.

Для получения ортогональной проекции вершины не требуется каких-либо дополнительных преобразований.

В моем проекте будет использоваться одноточечная перспективная проекция. Для получения перспективной проекции вершины необходимо умножить полученную на предыдущих шагах матрицу ее трансформации на матрицу одноточечной перспективной проекции:

## Отбрасывание нелицевых граней

С помощью отбрасывания нелицевых граней моделей при построении изображения можно существенно сократить временные затраты, так как грани, невидимые по отношению к камере, растеризоваться не будут.

Пусть

* – вектор внешней нормали к грани модели
* – вектор от камеры до любой точки грани

Для определения видимости грани используется формула:

## Преобразования трёхмерной сцены в пространство области изображения

Преобразования трёхмерной сцены в пространство области изображения предполагают масштабирование полученного изображения до приемлемого размера, а затем смещение его на центр экрана.

## Алгоритм Z-буфера

**Вход:** Грань модели

**Выход:** Обновленные буферы кадра и глубины

Общая схема алгоритма изображена на рисунке 6:

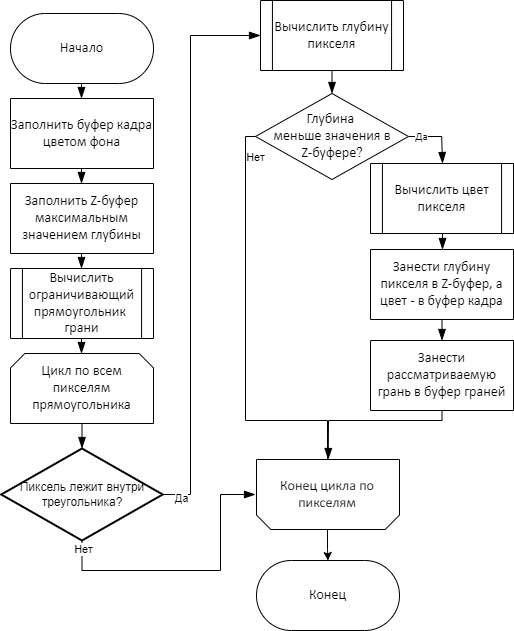


Рисунок 6. Алгоритм Z-буфера

Как видно из алгоритма выше, для растеризации треугольной грани сначала необходимо найти ограничивающий её прямоугольник. Таким образом можно уменьшить количество обрабатываемых пикселей. Затем для каждого пикселя ограничивающего прямоугольника находятся его барицентрические координаты относительно вершин треугольной грани.

Пусть

* A, B, C – вершины полигона,
* P – пиксель внутри ограничивающего прямоугольника.

Площадь треугольника можно найти по следующей формуле:

Тогда барицентрические координаты пикселя равны:

В случае, если хоть одна из барицентрических координат отрицательна, то пиксель лежит вне полигона. Если пиксель лежит внутри треугольника, то найти значение его глубины можно по следующей формуле:

## Простая модель освещения

В простом методе освещения цвет пикселя рассчитывается по закону Ламберта. При этом положение наблюдателя не имеет значения, т.к. диффузно отраженный свет рассеивается равномерно по всем направлениям (рисунок 7):

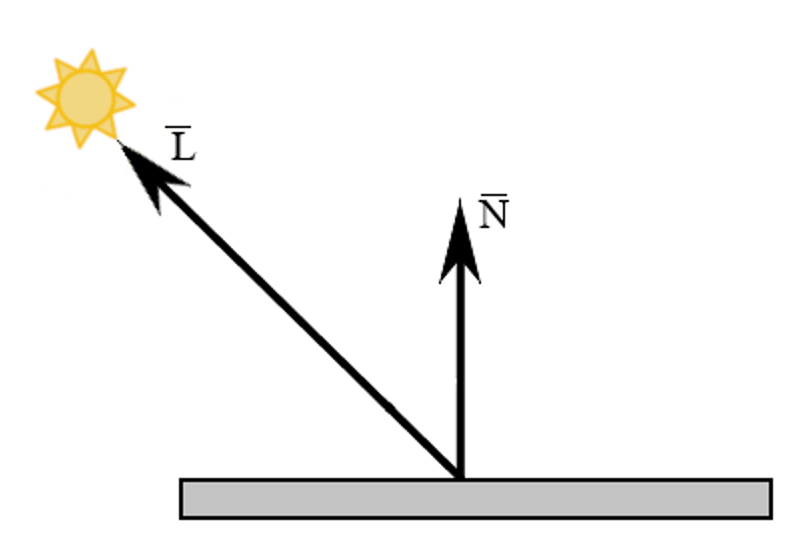


Рисунок 7. Закон Ламберта

Пусть

* I – результирующая интенсивность света в точке,
* – вектор от точки до источника,
* – вектор нормали,
* I0 – интенсивность источника,
* Kd – коэффициент диффузного освещения.

Формула расчёта интенсивности имеет следующий вид:

## Алгоритм выбора составляющей части модели

Общая схема алгоритма выбора составляющей части модели изображена на рисунке 8:

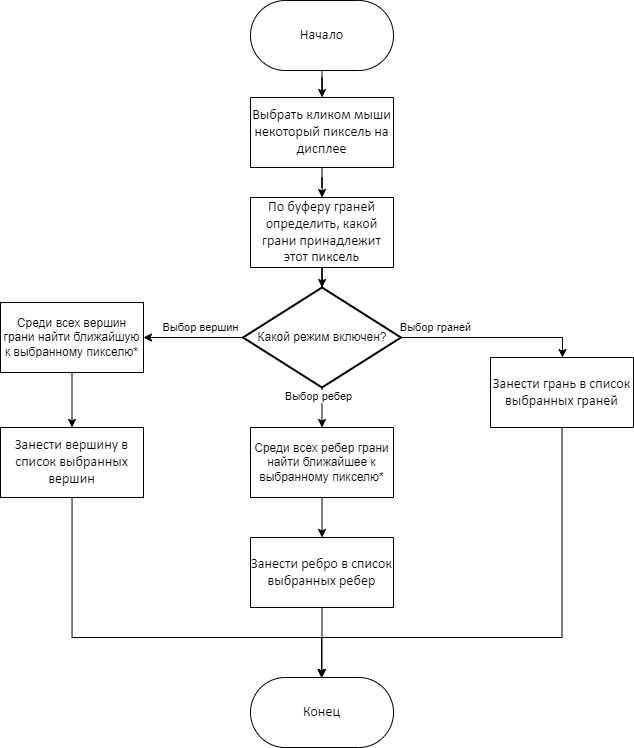


Рисунок 8. Выбор составляющей части модели

\* Для того, чтобы найти среди всех вершин ближайшую к выбранному пикселю, необходимо для начала высчитать экранные координаты этой вершины. Аналогичным способом происходит и поиск ближайшего ребра.

# Технологический раздел

## Выбор средств реализации

В качестве языка программирования был выбран С++, так как:

* Я ознакомился с этим языком в рамках курса по ООП
* Данный язык программирования поддерживает объектно-ориентированную модель разработки, что позволяет четко структурировать программу и легко модифицировать отдельные ее компоненты независимо от других.
* Язык C++ позволяет эффективно использовать ресурсы системы благодаря широкому набору функций и классов из стандартной библиотеки.

В качестве среды разработки была выбрана «Microsoft Visual Studio», так как:

* Для студентов это ПО распространяется бесплатно
* Поддерживает интеграцию с Qt
* Имеет встроенную технологию автодополнения IntelliSense, позволяющую сильно облегчить процесс разработки
* Имеет встроенный и удобный в использовании отладчик

В качестве графического фреймворка был выбран Qt, так как:

## 3.2 Структура и состав классов

Источник света

class LightSource

{

public Point3D position;

public Vector direction;

public Color color;

}

Модель

class Model

{

List<Point3D> vertices;

public List<Polygon> polygons;

private Color basicColor;

}

Система частиц

class ParticleSystem

{

List<Drop> system;

Vector direction;

int xMax, yMax;

int intensity;

}

class Drop

{

int x, y, z;

}

Математические абстракции

class Point3D

{

public int x, y, z;

}

class Vector

{

public double x, y, z;

public double length;

}

class Polygon

{

List<Point3D> vertices;

Color basicColor;

public List<Point3D> pointsInside;

Vector normal;

}

Класс обработки сцены

class Zbuffer

{

private Bitmap res;

private int[][] Zbuf;

}

## 3.3 Сведения о модулях программы

Program – главная точка входа в приложение;

Form1 – интерфейс;

Light – описание источников света;

Model – Описание объектов сцены;

Rain – Описание дождя;

Zbuffer – Алгоритм Z буфера.

## 3.4 Интерфейс программы

Группа «Сцена»

Позволяет добавить параллелепипед с заданными размерами и положением на сцену.

Группа «Источник света»

Позволяет выбрать направление источника света.

Группа «Направление ветра»

Позволяет задать направление ветра, который, в свою очередь, задает направление движения и длину капель.

Группа «Дождь»

Позволяет задать интенсивность дождя – количество капель, добавляемых при обновлении кадра.

Задержка (в мс) – задает частоту смены кадров.

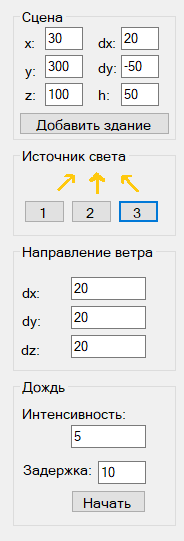


Рисунок 3 Интерфейс программы

# Заключение

Во время выполнения поставленной задачи были рассмотрены и проанализированы основные алгоритмы удаления невидимых линий, построения теней, методы закрашивания, методы генерации осадков. Были проанализированы их достоинства и недостатки, выбраны наиболее подходящие для решения поставленной задачи.

Разработанный программный продукт синтезирует трехмерное изображение при помощи алгоритмов компьютерной графики. Программа реализована таким образом, что пользователь может добавлять новые объекты на сцену, изменять характеристики ветра и дождя, изменять положение источника света.

В ходе выполнения поставленной задачи были изучены возможности Windows Forms, получены знания в области компьютерной графики.

# Список использованной литературы

1. Методы представления дискретных данных [Электронный ресурс]. – Режим доступа:

<https://www.graphicon.ru/oldgr/ru/library/multires_rep/index.html> (дата обращения 27.06.19)

1. Полигональная сетка [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D0%B3%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D0%BA%D0%B0> (дата обращения 27.06.19)
2. Е. А. Снижко. Компьютерная геометрия и графика [Текст], 2005. - 17 с.
3. Проблемы трассировки лучей – из будущего в реальное время. [Электронный ресурс]. – Режим доступа: <https://nvworld.ru/articles/ray-tracing/3/> (дата обращения 28.06.19)
4. RayTracing – царь света и теней, Лев Дымченко [Электронный ресурс]. – Режим доступа: <https://old.computerra.ru/206167/> (дата обращения 28.06.19)
5. Реалистичная визуализация атмосферных осадков в приложениях реального времени, В. В. Карабчевский, А.А. Лунтовская, Донецк ДонНТУ [Текст], 2015
6. K. Garg, S. K. Nayar. Photorealistic rendering of rain streaks. In ACM SIGGRAPH 2006 Papers. SIGGRAPH '06. ACM, New York, NY, 996‐ 1002 с.