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»

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ
НА ТЕМУ
«Реализация программы передвижения по закрытым комнатам»

Студент ИУ7-52Б _____ Сучков А.Д.
(Группа) (подпись, дата) (фамилия, и.о.)

Руководитель курсового проекта _____ Кострицкий А. С.
(подпись, дата) (фамилия, и.о.)

2020 г.

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

УТВЕРЖДАЮ
Заведующий кафедрой ___ИУ7____
(Индекс)
_____ _И.В. Рудаков_
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

ЗАДАНИЕ
на выполнение курсового проекта

по дисциплине _____Компьютерная графика_____

Студент группы _____ИУ7-42Б_____

Сучков Александр Дмитриевич
(Фамилия, имя, отчество)

Тема курсового проекта __”Реализация программы передвижения по закрытым комнатам”

_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____учебный_____

Источник тематики (кафедра, предприятие, НИР) __кафедра_____

График выполнения проекта: 25% к _3_ нед., 50% к _9_ нед., 75% к 12 нед., 100% к _15_ нед.

Задание

Разработать программу для построения трёхмерного изображения, которое наполнено статичными объектами - элементами декора, и динамичными элементами – моделями в виде спрайтов. Пользователем задаются количество, расположение и геометрия элементов декора, количество и положение моделей-спрайтов, а также количество, мощность и расположение источников света. Реализовать взаимодействие пользователя с подвижной камерой, с помощью которой будет производится перемещение по уровню. Предусмотреть наличие источника света на бесконечности.

Оформление курсового проекта:

Расчетно-пояснительная записка на 25-35 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.): на защиту проекта должна быть представлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс, характеристики разработанного ПО, результаты проведенных исследований.

Дата выдачи задания « ___ » _____ 20__ г.

Руководитель курсового проекта _____ __А.С. Кострицкий__
(Подпись, дата) (И.О.Фамилия)

Студент _____ ___А.Д. Сучков_____
(Подпись, дата) (И.О.Фамилия)

Оглавление

Введение

В настоящее время компьютерная графика имеет достаточно широкий охват применимости во многих отраслях нашей жизни: для визуализации данных на производстве, для создания реалистичных специальных эффектов в кино и для создания видеоигр.

Из-за этого перед специалистами, создающими трёхмерные изображения, встаёт множество трудностей связанных с учётом таких явлений, как преломление, отражение и рассеивание света, а для достижения максимально реалистичного изображения также учитываются дифракция, интерференция, вторичные отражения света, цвет и текстура объектов.

В компьютерной графике существует множество разнообразных алгоритмов, помогающих решить эти задачи, однако большинство являются достаточно ресурсоёмкими, так как чем более реалистичное изображение мы хотим получить, тем больше нам необходимо времени и памяти на синтез. Это, пожалуй, самая главная проблема при создании реалистичных изображений (особенно динамических сцен), которую пытаются решить и по сей день.

Создание видеоигр крепко вплелось в индустрию развлечений. На данный момент существует множество больших компаний, состоящих из огромного количества специалистов, которые соревнуются друг с другом в достижении наивысшего реализма и качества графики. Именно в этой отрасли можно наглядно видеть эволюцию в компьютерной графике, как с каждым годом создаётся всё больше игр, использующих сложные технологии и алгоритмы визуализации.

Целью курсового проекта является разработка программы для визуализации трёхмерной сцены, которое наполнено статичными объектами – элементами декора, динамичными элементами – моделями в виде спрайтов и источниками света.

В рамках реализации проекта должны быть решены следующие задачи:

- изучение и анализ алгоритмов компьютерной графики, использующихся для создания реалистичной модели взаимно перекрывающихся объектов, и выбор наиболее подходящего для решения поставленной задачи;
- проектирование архитектуры программного обеспечения;
- реализация выбранных алгоритмов и структур данных;
- разработка программного обеспечения, которое позволит отобразить и собрать трехмерную сцену;
- разработка игровой механики в виде подвижной камеры от первого лица.

Итогом данной работы является программное обеспечение, которое позволяет построить собственные сцены и уровни для игры. В программе реализованы подвижная камера от первого лица с возможностью поворотов по двум осям, система загрузки собственных трёхмерных моделей или уровней из файлов obj формата, их передвижение и масштабирование, а также конструктор сцены из подгруженных объектов.

1. Аналитическая часть

В данном части проводится анализ объектов сцены и существующих алгоритмов построения изображений и выбор более подходящих алгоритмов для дальнейшего использования.

1.1 Постановка задачи

Необходимо обеспечить отрисовку трёхмерной сцены, наполненной различными объектами и спрайтами, учитывая освещение от находящихся там точечных источников света. Данная программа является игрой и может использоваться в качестве развлечения, а также с реализованной механикой построения сцен, может использоваться в разработке дизайна локаций.

1.2 Описание и формализация объектов сцены

Для описания трёхмерных геометрических объектов существует три модели: каркасная, поверхностная и объёмная. Для реализации поставленной задачи, более подходящей моделью будет поверхностная, так как, по сравнению с каркасной, она может дать более реалистичное и понятное зрителю изображение. В то же время, объёмной модели требуется больше памяти и поэтому она будет скорее излишняя, в условиях моей задачи.

В свою очередь поверхностная модель может задаваться параметрическим представлением или полигональной сеткой.

При параметрическом представлении поверхность можно получить при вычислении параметрической функции, что очень удобно при просчёте поверхностей вращения, однако, ввиду их отсутствия, данное представление будет не выгодно.

В случае полигональной сетки форма объекта задаётся некоторой совокупностью вершин, рёбер и граней, что позволяет выделить несколько способов представления:

1. Вершинное представление – при таком представлении вершины хранят указатели на соседние вершины. В таком случае для

рендеринга нужно будет обойти все данные по списку, что может занимать достаточно много времени при переборе.

2. Список граней – при таком представлении объект хранится, как множество граней и вершин. В таком случае достаточно удобно производить различные манипуляции над данными.
3. Таблица углов – при таком представлении вершины хранятся в предопределённой таблице, такой что обход таблицы неявно задаёт полигоны. Такое представление более компактное и более производительное для нахождения полигонов, однако операции по замене достаточно медлительны.

Наиболее подходящим представлением сцены в условиях поставленной задачи будет представление в виде списка граней, т.к. оно позволяет эффективно манипулировать данными, а также позволяет проводить явный поиск вершин грани и самих граней, которые окружают вершину.

Сцена состоит из следующих объектов:

- трёхмерные объекты – объекты, имеющие произвольную форму, которые заданы файлами в формате obj;
- спрайты – подвижные объекты, которые представлены в виде плоскости;
- источники света – материальная точка в пространстве, из которой исходят лучи света во все стороны. Положение на сцене регулируется координатами (x, y, z).

1.3 Критерии выбора алгоритмов

Важным критерием выбора алгоритма служит быстродействие отрисовки и обновления сцены.

1.4 Анализ алгоритмов удаления невидимых линий и поверхностей

Выбирая подходящий алгоритм удаления невидимых линий, необходимо учитывать поставленную задачу и её особенности, а именно то, что сцена должна быть динамической, а следовательно, алгоритм должен быть

достаточно быстрым. Также, стоит заметить, что в реализации подобной игры DOOM 1993 года использовались BSP деревья для построения корректного изображения. В настоящее время существует множество различных алгоритмов построения изображений: алгоритм трассировки лучей, алгоритм Варнока, алгоритм Робертса, алгоритм Z-буфера и т.д. Рассмотрим преимущества и недостатки каждого из алгоритмов.

BSP-деревья

Двоичное разбиение пространства – метод рекурсивного разбиения евклидового пространства в выпуклые множества и гиперплоскости. В результате объекты сцены хранятся в виде структуры данных, называющаяся BSP-деревом. Каждый узел дерева связан с разбивающей плоскостью, при этом все объекты, которые лежат с фронтальной стороны плоскости, относятся к фронтальному поддереву, а все объекты с другой стороны, относятся к оборотному поддереву. Для определения положения объекта анализируются положения каждой его точки.

Данный метод достаточно эффективен в:

- сортировке визуальных объектов в порядке удаления от наблюдателя;
- обнаружении столкновений.

К недостатку можно отнести то, что для достижения высокой производительности в программе, алгоритм необходимо оптимизировать, что достаточно сильно может повлиять на время и сложность разработки.

Алгоритм обратной трассировки лучей

Алгоритм является улучшенной модификацией алгоритма прямой трассировки. Из камеры испускаются лучи, проходящие через каждый пиксель вглубь сцены, затем идет поиск пересечений первичного луча с объектами сцены, как показано на рисунке 1.1, в случае обнаружения пересечения, рассчитывается интенсивность пикселя, в зависимости от положения источника света, при отсутствии пересечения, пиксель закрашивается цветом фона.

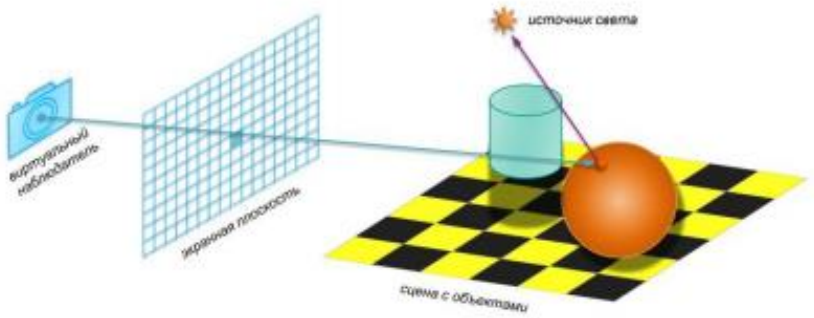

Рисунок 1.1. Работа алгоритма обратной трассировки лучей.

К достоинствам данного алгоритма можно отнести возможность получения изображения гладких объектов без аппроксимации их примитивами (например, треугольниками). Трассировка лучей позволяет визуализировать тени, эффекты прозрачности, преломления, отражения. Вычислительная сложность метода линейно зависит от сложности сцены. Полученное изображение получается достаточно реалистичным.

Однако, недостатком алгоритма является его производительность. Для получения изображения необходимо создавать большое количество лучей, проходящих через сцену и отражаемых от объекта. Это приводит к существенному снижению скорости работы программы.

В поставленной мною задаче не используются явления отражения и преломления света и поэтому некоторые вычисления окажутся излишними, кроме того, скорость синтеза сцены должна быть высокой, из-за чего алгоритм обратной трассировки лучей не подходит.

Алгоритм Варнока

Преимуществом данного алгоритма является то, что он работает в пространстве изображений. Алгоритм предлагает разбиение области рисунка на более мелкие окна, и для каждого такого окна определяются связанные с ней многоугольники и те, видимость которых «легко» определить, изображаются на сцене.

К недостаткам можно отнести то, что в противном случае разбиение повторяется, и для каждой из вновь полученных подобластей рекурсивно применяется процедура принятия решения. По предположению, с уменьшением размеров области, её будет перекрывать всё меньшее количество многоугольников. Считается, что в пределе будут получены области, которые содержат не более одного многоугольника, и решение будет принято достаточно просто.

Из-за того, что поиск может продолжаться до тех пор, пока либо остаются области, содержащие не один многоугольник, либо пока размер области не станет совпадать с одним пикселом, алгоритм не подходит под условия моей задачи.

Алгоритм Робертса

Алгоритм Робертса представляет собой первое известное решение задачи об удалении невидимых линий. Этот алгоритм работает в объектном пространстве. Сначала удаляются из каждого тела те ребра или грани, которые перекрываются самим телом. Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части, если таковые есть, перекрываются этими телами.

Данный алгоритм обладает достаточно простыми и одновременно мощными математическими методами, что, несомненно, является преимуществом. Некоторые реализации, например использующие предварительную сортировку по оси z, могут похвастаться линейной зависимостью от числа объектов на сцене.

Однако алгоритм обладает недостатком, который заключается в вычислительной трудоёмкости, растущей теоретически, как квадрат числа объектов, что может негативно сказаться на производительности. В то же время реализация оптимизированных алгоритмов достаточно сложна.

Алгоритм, использующий Z буфер

Данный алгоритм удаления невидимых поверхностей является одним из самых простых и широко используемых. Этот алгоритм работает в пространстве изображения. Его идея заключается в использовании двух буферов: буфера кадра и буфера глубины, также называемого Z-буфером. Буфер кадра используется для хранения интенсивности каждого пикселя в пространстве изображения. В буфере глубины запоминается значение координаты Z (глубины) каждого видимого пикселя в пространстве изображения. 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. На рисунке 1.2 представлен пример работы z-буфера.

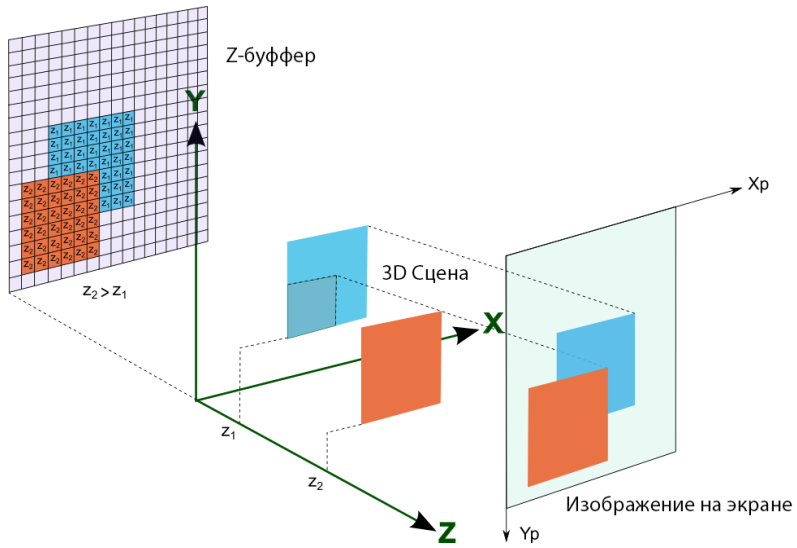

Рисунок 1.2. Работа алгоритма z-буфера.

Главными преимуществами данного алгоритма являются его достаточная простота реализации и функциональность, которая более чем подходит для визуализации динамической сцены игры. Также алгоритм не тратит время на сортировку элементов сцены, что даёт значительный прирост к производительности.

К недостатку алгоритма можно отнести большой объём памяти необходимый для хранения информации о каждом пикселе. Однако данный недостаток является незначительным ввиду того, что большинство современных компьютеров обладает достаточно большим объёмом памяти для корректной работы алгоритма.

1.5 Выбор модели освещения

На сегодняшний день существует две модели освещения, которые используются для построения света на трёхмерных сценах: локальная и глобальная. Физические модели, которые не учитывают перенос света между поверхностями (не используют вторичное освещение), называются локальными. В противном случае модели называются глобальными или моделями глобального освещения.

Исходя из поставленной задачи, нет необходимости использовать глобальные модели освещения, так как важна производительность, а изображение не должно быть реалистичным.

Далее в разделе будут рассмотрены только локальные модели освещения.

Модель Ламберта

Данная модель является одной из самых простых моделей освещения. Модель Ламберта моделирует идеальное диффузное освещение, идея в том, что свет, падающий в точку, одинаково рассеивается по всем направлениям полупространства. Сила освещения зависит исключительно от угла α между

вектором падения света L и вектором нормали N, как показано на рисунке 1.3. Максимальная сила света будет при перпендикулярном падении света на поверхность и будет убывать с увеличением угла α.

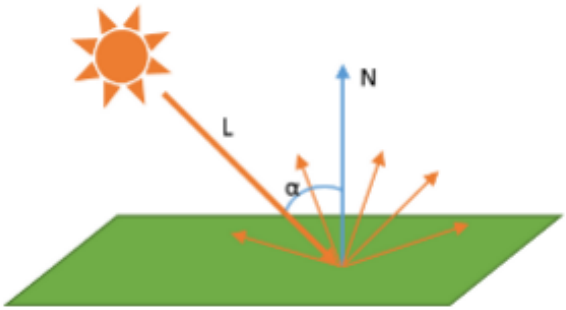

Рисунок 1.3. Модель освещения Ламберта.

Модель достаточно проста, с её помощью невозможно передать блики на объектах сцены и за счёт этого можно добиться высокой производительности.

Модель Фонга

Идея такой модели заключается в предположении, что освещённость каждой точки тела разлагается на 3 компоненты (см. рис 1.4): фоновое освещение (ambient), рассеянный свет (diffuse), бликовая составляющая (specular).

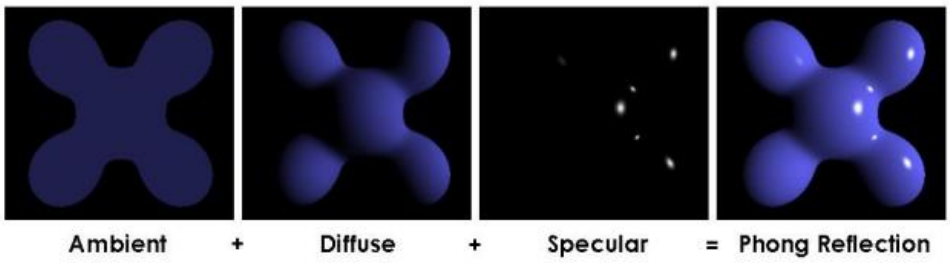

Рисунок 1.4. Модель освещения Фонга.

Свойства источника определяют мощность излучения для каждой из этих компонент, а свойства материала поверхности определяют её способность воспринимать каждый вид освещения. Фоновое освещение присутствует в

любом уголке сцены и никак не зависит от каких-либо источников света, поэтому для упрощения расчетов оно задается константой. Диффузное освещение рассчитывается аналогично модели Ламберта. Отраженная составляющая освещенности (блики) в точке зависит от того, насколько близки направления вектора, направленного на наблюдателя (вектор V на рисунке 1.5), и отраженного луча (вектор R на рисунке 1.5).

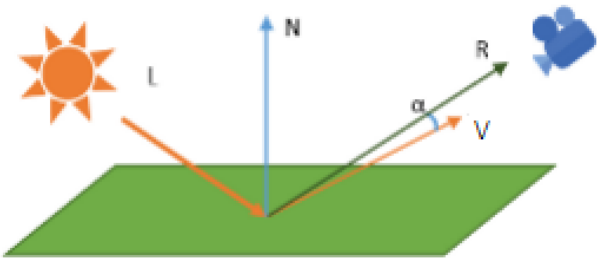

Рисунок 1.5. Получение бликов в модели освещения Фонга.

Формула расчёта:

$$I = K_a I_a + K_d(\vec{N}, \vec{L}) + K_s(\vec{R}, \vec{V})^p$$, где

$\vec{N}$ – вектор нормали к поверхности в точке,

$\vec{L}$ – падающий луч (направление на источник света),

$\vec{R}$ – отраженный луч,

$\vec{V}$ – вектор, направленный к наблюдателю,

K_a – коэффициент фонового освещения,

K_d – коэффициент диффузного освещения,

K_s – коэффициент зеркального освещения.

p – степень, аппроксимирующая пространственное распределение зеркально отраженного света.

При этом всём, все векторы являются единичными.

Данная модель освещения достаточно сильно улучшает визуальные качества сцены, по сравнению с вышеописанной моделью Ламберта, добавляя в неё блики, однако может достаточно сильно нагрузить программу.

1.6 Анализ алгоритмов закрашивания

Алгоритм простой закраски

По закону Ламберта, вся грань закрашивается одним уровнем интенсивности. Метод является достаточно простым в реализации и не требовательным к ресурсам. При этом алгоритм плохо учитывает отражения и при отрисовки тел вращения, возникают проблемы.

Алгоритм закраски по Гуро

В основу данного алгоритма положена билинейная интерполяция интенсивностей, позволяющая устранять дискретность изменения интенсивности. Благодаря этому криволинейные поверхности будут более гладкими (см. рис 1.6).

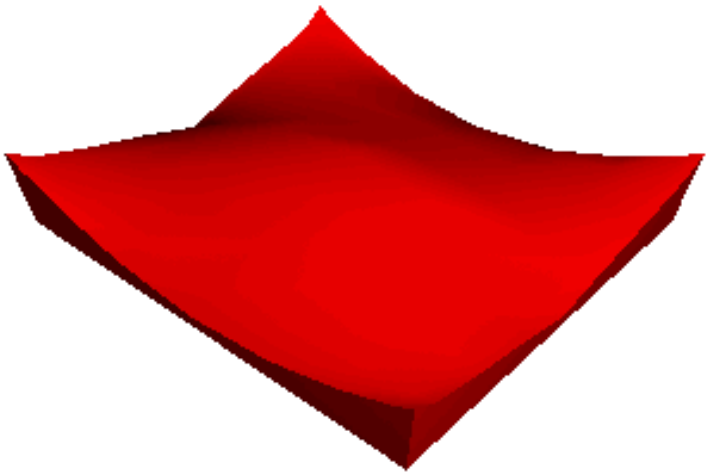

Рисунок 1.6. Пример закраски по Гуро.

Алгоритм закраски по Фонгу

В данном алгоритме за основу берётся билинейная интерполяция векторов нормалей, благодаря чему достигается лучшая локальная аппроксимация кривизны поверхности и, следовательно, изображение выглядит более реалистичным.

Однако алгоритм требует больших вычислений, по сравнению с алгоритмом по Гуро, так как происходит интерполяция значений векторов нормалей.

Вывод

В результате анализа алгоритмов, опираясь на поставленную задачу, были выбраны следующие алгоритмы.

1. Алгоритм, использующий Z буфер, является наиболее подходящим для построения динамической игровой сцены.
2. Локальная модель освещения Ламберта, так как программа не должна выводить реалистичного изображения и должна иметь наиболее высокую производительность.
3. Закраска по Гуро, так как алгоритм достаточно быстр, и он хорошо сочетается с выбранным ранее Z буфером.

2. Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов для Z буфера и закраски по Гуро. Также будет приведена возможная оптимизация алгоритмов, геометрические преобразования для подвижной камеры и диаграмма классов.

Программа должна обладать следующей функциональностью:

1. Визуализировать трёхмерную сцену, состоящую из объектов, представленных в пункте 1.2 аналитической части, в режиме реального времени.
2. Предоставить в интерфейсе возможность пользователю выполнять следующие действия:

   2.2 Добавлять объекты на сцену и подгружать уровни.

   2.3 Изменять параметры объектов сцены.

   2.4 Изменять положение камеры и осуществлять её поворот.

   2.5 Добавлять точечные источники света.

   2.6 Изменять положение и параметры источников света.

2.1 Схемы алгоритмов

На рисунке 2.1 представлена схема алгоритма Z буфера.

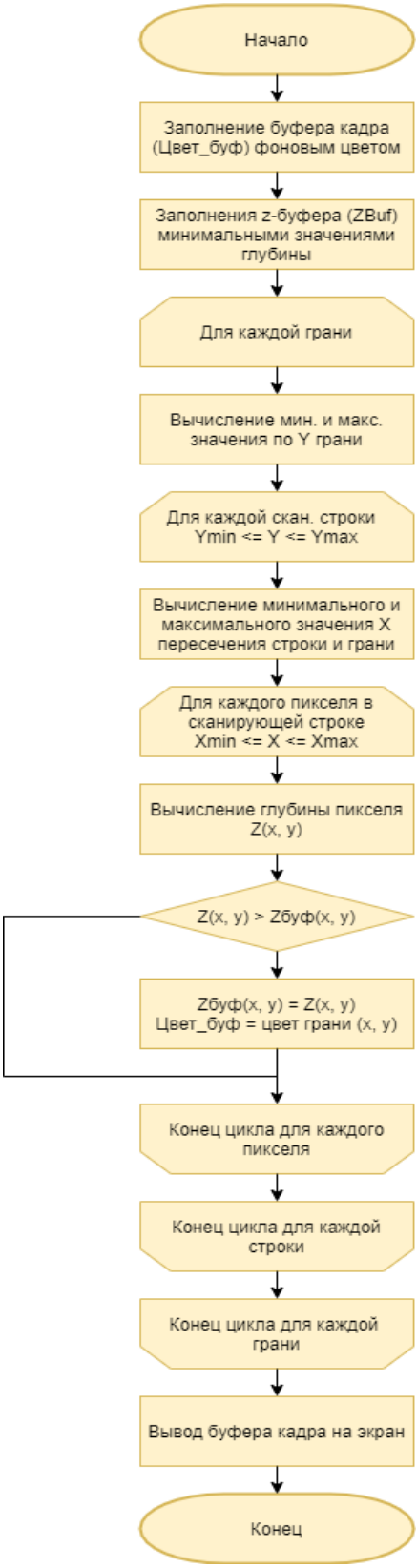

Рисунок 2.1 Схема алгоритма Z буфера.

На рисунке 2.2 представлена схема алгоритма закраски по Гуро.

Рисунок 2.2 Схема алгоритма Z закраски по Гуро.

2.2 Алгоритм Z-буфера

Данный алгоритм является одним из простейших алгоритмов удаления невидимых поверхностей. Впервые он был предложен Кэтмулом. Алгоритм работает в пространстве изображения, а сама идея z-буфера является простым обобщением идеи о буфере кадра, который используется для запоминания атрибутов или интенсивности каждого пиксела в пространстве изображения.

Z-буфер – это отдельный буфер глубины, который используется для запоминания координаты z или глубины каждого видимого пиксела в пространстве изображения. В процессе работы глубина или значение z каждого нового пиксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесён в z-буфере.

Если это сравнение показывает, что новый пиксел расположен впереди пиксела, который находится в буфере кадра, то новое значение заносится в буфер и, кроме этого, производится корректировка z-буфера новым значением z. Если при сравнении получается противоположный результат, то никаких действий не производится.

По сути, алгоритм представляет из себя поиск по x и y наибольшего значения функции z(x, y). Формальное описание алгоритма z-буфера:

1. Заполнить буфер кадра фоновым значением интенсивности или цвета
2. Провести инициализацию Z-буфера минимальным значением глубины
3. Преобразовать каждый многоугольник в растровую форму в произвольном порядке

    3.1 Для всех пикселей, которые связаны с многоугольником, вычислить его глубину z(x, y)

3.2 Глубину пиксела сравнить со значением, которое хранится в буфере: если $z(x, y) > z_{буф}(x, y)$, то $z_{буф}(x, y) = z(x, y)$, $цвет(x, y) = цвет_{пикселя}$

4. Отобразить результат

2.3 Модель освещения

Для достижения наиболее высокой производительности, будем считать в локальной модели освещения две составляющие интенсивности, а именно фоновую освещённость (I_{amb}) и диффузное отражение (I_{diff}).

Фоновое освещение это постоянная в каждой точке величина надбавки к освещению. Вычисляется следующим образом:

$$I_{amb} = k_a \cdot I_a,$$

где I_{amb} – интенсивность отражённого Ambient-освещения, k_a – коэффициент в пределах от 0 до 1, характеризующий отражающие свойства поверхности для Ambient-освещения, I_a – исходная интенсивность Ambient-освещения, которое падает на поверхность.

$$I_{diff} = I_d \cdot k_{diff} \cdot \cos(\theta),$$

где I_d – интенсивность падающего на поверхность света, k_{diff} – коэффициент в пределах от 0 до 1, характеризующий рассеивающие свойства поверхности, $\cos(\theta)$ – угол между направлением на источник света и нормалью поверхности.

С учётом только фоновой освещённости, можно построить изображение (см. рис 2.1).

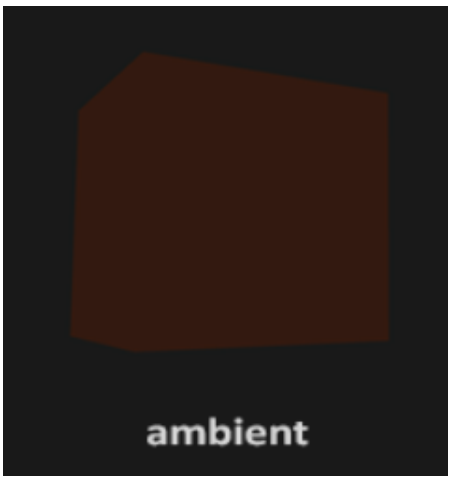

Рисунок 2.1 Изображение с учётом только фоновой освещённости.

С учётом и фоновой освещённости, и диффузного отражения, получим более реалистичное изображение (см. рис 2.2).

Рисунок 1.2 Изображение с учётом фоновой и диффузной освещённостей.

Тогда, конечная формула принимает вид:

$$I = I_{amb} + I_{diff} = k_a \cdot I_a + I_d \cdot k_{diff} \cdot \cos(\theta)$$

2.4 Алгоритм Гуро

Данный метод закраски, основанный на интерполяции интенсивности, позволяет устранять дискретность изменения интенсивности.

Из рисунка 2.3 можно выделить формулу нормалей:

$$\overline{N}_a = (\overline{N}_1 + \overline{N}_2 + \overline{N}_3) / 3$$

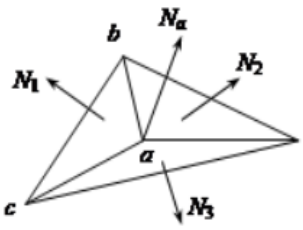

Рисунок 2.3. Нормали к вершинам.

Алгоритм делится на четыре этапа:

1. Вычисляются нормали к каждой грани.
2. Определяются нормали в вершинах путём усреднения нормалей по всем полигональным граням, которым принадлежит вершина.
3. Используя нормали в вершинах и применяя произвольный метод закраски, вычисляются значения интенсивности в вершинах.
4. Каждый многоугольник закрашивается путём линейной интерполяции значений интенсивностей в вершинах сначала вдоль каждого ребра, а затем и между рёбрами вдоль каждой сканирующей строки (рис. 2.4).

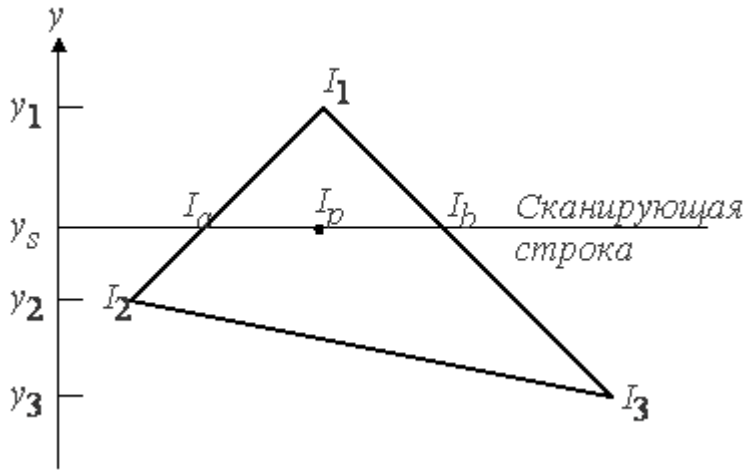

Рисунок 2.4 Линейная интерполяция интенсивностей.

Интерполяцию вдоль рёбер легко объединить с алгоритмом удаления скрытых поверхностей, которое построено на принципе построчного сканирования. Для всех рёбер запоминается начальная интенсивность, а также изменение интенсивности при каждом единичном шаге по координате y.

Заполнение видимого интервала на сканирующей строке производится путём интерполяции между значениями интенсивности на двух рёбрах, которые ограничивают интервал.

$$I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2};$$

$$I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3};$$

$$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}.$$

Для цветных объектов отдельно интерполируется каждая из компонент цвета.

2.5 Оптимизация алгоритмов

Работу некоторых алгоритмов можно объединить, чтобы добиться более производительной работы. Алгоритм z-буфера можно объединить с алгоритмом Гуро, так как они оба используют интерполяцию, также в функцию z-буфера можно занести сравнение с буфером тени.

Чтобы добиться сокращения количества вычислений при проецировании точек из буфера тени на z-буфер, можно найти минимальный и максимальный пиксел, глубина которого отлична от максимальной глубины. Это позволит сократить итерации цикла.

2.6 Геометрические преобразования

Для осуществления поворота и перемещения камеры используются смены базиса в трёхмерном пространстве.

В евклидовом пространстве система координат (репер) задаётся точкой отсчёта и базисом пространства. Тогда вектор OP в репере (O, i, j, k) задаётся следующим образом:

$$\overrightarrow{OP} = \vec{i}x + \vec{j}y + \vec{k}z = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Допустим, что существует второй репер (O`, i` j` k`) и необходимо преобразовать координаты точки, данные в одном репере, в другой репер. Можно заметить, что (i, j, k) и (i`, j`, k`) – это базисы, то существует невырожденная матрица M, такая, что:

$$\begin{bmatrix} \vec{i'} & \vec{j'} & \vec{k'} \end{bmatrix} = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \times M$$

Для наглядности можно построить рисунок 2.5.

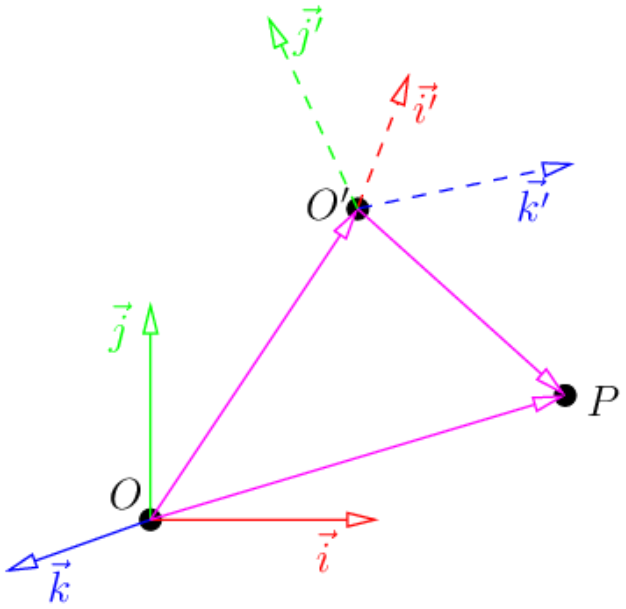

Рисунок 2.5 Преобразование координат точки P.

Далее можно расписать представление вектора OP:

$$\overrightarrow{OP} = \overrightarrow{OO'} + \overrightarrow{O'P} = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \begin{bmatrix} O'_x \\ O'_y \\ O'_z \end{bmatrix} + \begin{bmatrix} \vec{i'} & \vec{j'} & \vec{k'} \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

Подставляя во вторую часть выражение замены базиса, получим:

$$\overrightarrow{OP} = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \left(\begin{bmatrix} O'_x \\ O'_y \\ O'_z \end{bmatrix} + M \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \right)$$

Из вышеописанных формул можно получить формулу замены координат для двух базисов:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} O'_x \\ O'_y \\ O'_z \end{bmatrix} + M \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = M^{-1} \left(\begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} O'_x \\ O'_y \\ O'_z \end{bmatrix} \right)$$

Как следствие, если нам необходимо подвинуть камеру, то сдвигается вся сцена, оставляя камеру неподвижной.

Допустим, необходимо, чтобы камера находилась в точке e (eye), смотрела в точку c (center) и чтобы заданный вектор u (up) на финальном изображении был бы вертикален (см. рис 2.6).

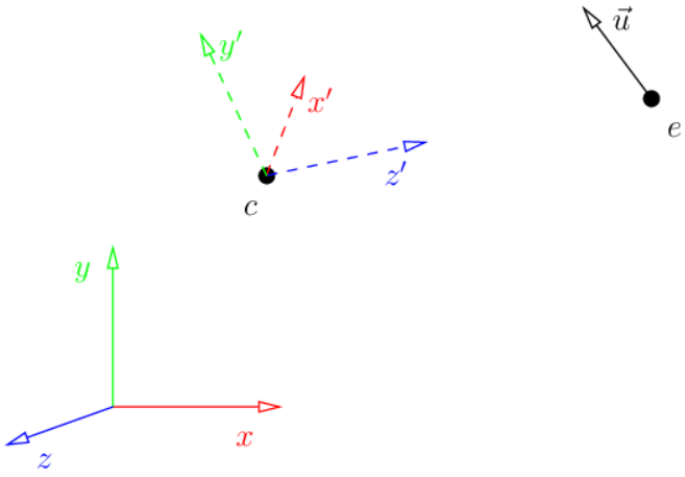

Рисунок 2.6 Построение финального изображения.

Тогда, рендер будет строится в репере (c, x`y`z`). Модель задана в репере (O, xyz), значит, необходимо посчитать репер x`y`z` и соответствующую матрицу перехода по вышеописанным формулам.

2.7 Диаграмма классов

На рисунке 2.7 приведена диаграмма классов.

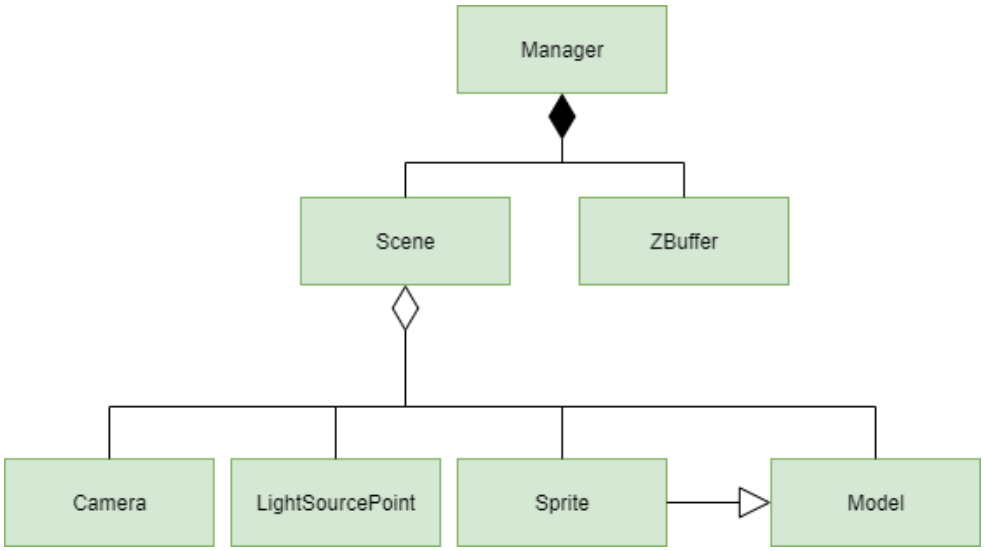

Рисунок 2.7 Схема классов программы.

Разработанная программа состоит из следующих классов:

Классы объектов сцены

- Model – класс трёхмерных объектов с возможностью перемещения, масштабирования и поворота вокруг собственного центра.
- Sprite – класс динамических объектов – спрайтов с возможностью определения конечной точки и скорости для перемещения.
- Camera – класс камеры с возможностью перемещения.
- LightSourcePoint – класс источника освещения с возможностью перемещения по сцене и изменения мощности.

Вспомогательные классы сцены

- Scene – контейнер, который содержит в себе объекты сцены.

- ZBuffer – контейнер, который предоставляет удобный функционал для работы с буфером глубины.

Класс отрисовки сцены

- Drawer – класс менеджер, который связывает сцену и интерфейс системы.

Вывод

В данном разделе были рассмотрены схемы алгоритмов, их разбор, оптимизация и диаграммы классов.

3.Технологическая часть

В данном разделе рассмотрены выбор средств реализации и интерфейс программы, описаны основные этапы программной реализации.

3.1 Выбор языка программирования и среды разработки

Для реализации проекта в качестве языка программирования был выбран C++ — компилируемый статически типизированный язык программирования общего назначения. Поддерживает процедурное, объектно-ориентированное и обобщённое парадигмы программирования, также обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает в себя некоторые уже готовые контейнеры и алгоритмы.

C++ в себе сочетает свойства как высокоуровневых, так и низкоуровневых языков программирования. Также имеется возможность выполнять ассемблерные вставки – возможность встраивать низкоуровневый код, написанный на ассемблере, что может пригодится при оптимизации программы и увеличении её быстродействия. Язык широко используется для разработки программного обеспечения, область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также и для создания видеоигр.

Существует множество коммерческих и бесплатных реализаций языка C++ для различных платформ. К примеру GCC, Visual C++, Intel C++, Compiler.

Средой разработки была выбрана «Qt Creator» из-за того, что она бесплатна в пользовании студентами, имеет множество удобств, связанных с написанием и отладкой кода, позволяет достаточно быстро подключать и настраивать сторонние библиотеки. В качестве платформы выбрана OS Windows.

3.2 Описание основных этапов реализации

На рисунке 3.1 представлен код метода objectProcessing, который преобразовывает мировые координаты вершин модели в репер камеры, затем осуществляет перспективное искажение. Также метод вызывает функцию просчёта интенсивности и процедуру растеризации полигона.

```
void Drawer::objectProcessing(Model& model, Vector3f& camPos, Vector3f& camDir, Vector3f&
camUp)
{
    size_t i, j;
    bool skip;

    Vector3f center = model.getCenter();
    size_t faces    = model.getFacesCount();
    QColor color    = model.getColor();

    //Transformation matrix (not sure how it works)
    Matrix viewPort   = Camera::viewport(w/8, h/8, w*3/4, h*3/4);
    Matrix projection = Matrix::identity(4);
    Matrix modelView  = Camera::lookAt(camPos, camDir, camUp);

    projection[3][2]  = - 1.f / (camPos - camDir).norm();

    Matrix mvp = viewPort * projection * modelView;

    for (i = 0; i < faces; i++)
    {
        skip = false;
        std::vector<int> face = model.face(i);

        Vector3i screenCoords[3];
        float    intensity[3] = { BG_LIGHT, BG_LIGHT, BG_LIGHT };

        for (j = 0; j < 3; j++)
        {
            Vector3f v = center + model.vert(face[j]);

            if (v.z > camPos.z)
            {
                skip = true;
                break;
            }

            screenCoords[j] = Vector3f(mvp * Matrix(v));
            intensity[j] = lightProcessing(v, model.norm(i, j));
        }

        if (skip) continue;

        triangleProcessing(screenCoords[0], screenCoords[1], screenCoords[2],
                color, intensity[0], intensity[1], intensity[2]);
    }
}
```

Рисунок 3.1. Код метода objectProcessing.

На рисунке 3.2 представлен код метода lightProcessing, который просчитывает интенсивность вершины от источников света на сцене.

```
float Drawer::lightProcessing(const Vector3f& vert, const Vector3f& norm)
{
    float wholeIntensity = 0;
    float intensity;

    size_t lights = scene.getLightSourceCount();

    for (size_t i = 0; i < lights; i++)
    {
        intensity = 0;
        LightSourcePoint lsp = scene.getLightSource(i);

        Vector3f lightDir = vert - lsp.getPosition();

        intensity += lightDir * norm / pow(lightDir.norm(), 2.0);
        intensity *= lsp.getIntensity() * LIGHT_REFLECT;

        intensity = fmax(0.0, intensity);
        intensity = fmin(1.0, intensity);

        intensity = BG_LIGHT + intensity * (1 - BG_LIGHT);

        wholeIntensity += intensity;
    }

    if (wholeIntensity == 0)
        wholeIntensity = BG_LIGHT;
    else
        wholeIntensity /= lights;

    return wholeIntensity;
}
```

Рисунок 3.2. Код метода lightProcessing.

На рисунке 3.3 представлен метод triangleProcessing, который осуществлят вычисление глубины каждой точки полигона, а также её интенсивности.

```
void Drawer::triangleProcessing(Vector3i& t0, Vector3i& t1, Vector3i& t2,
                                const QColor& color, float& i0, float& i1, float& i2)
{
    if (t0.y == t1.y && t0.y == t2.y)
        return;

    if (t0.y > t1.y)
    {
        std::swap(t0, t1);
        std::swap(i0, i1);
    }
    if (t0.y > t2.y)
    {
        std::swap(t0, t2);
        std::swap(i0, i2);
```

```
    }

    if (t1.y > t2.y)
    {
        std::swap(t1, t2);
        std::swap(i1, i2);
    }

    int total_height = t2.y - t0.y;

    for (int i = 0; i < total_height; i++)
    {
        bool second_half = i > t1.y - t0.y || t1.y == t0.y;
        int segment_height = second_half ? t2.y - t1.y : t1.y - t0.y;

        float alpha = (float)i / total_height;
        float betta = (float)(i - (second_half ? t1.y - t0.y : 0)) / segment_height;

        Vector3i A =               t0 + Vector3f(t2 - t0) * alpha;
        Vector3i B = second_half ? t1 + Vector3f(t2 - t1) * betta :
                                   t0 + Vector3f(t1 - t0) * betta;

        float iA =               i0 + (i2 - i0) * alpha;
        float iB = second_half ? i1 + (i2 - i1) * betta : i0 + (i1 - i0) * betta;

        if (A.x > B.x)
        {
            std::swap(A, B);
            std::swap(iA, iB);
        }

        A.x = std::max(A.x, 0);
        B.x = std::min(B.x, w);

        for (int j = A.x; j <= B.x; j++)
        {
            float phi = B.x == A.x ? 1. : (float)(j - A.x) / (float)(B.x - A.x);

            Vector3i P = Vector3f(A) + Vector3f(B - A) * phi;
            float iP = iA + (iB - iA) * phi;

            if (P.x >= w || P.y >= h || P.x < 0 || P.y < 0) continue;

            if (zBuffer.getDepth(P.x, P.y) < P.z)
            {
                zBuffer.setDepth(P.x, P.y, P.z);
                colorCache[P.x][P.y] = QColor(iColor(color.rgba(), iP));
            }
        }
    }
}
```

Рисунок 3.3. Код метода triangleProcessing.

3.3 Описание интерфейса программы

На рисунке 3.4 представлен интерфейс основного окна программы в режиме игры.

Рисунок 3.4 Интерфейс основного окна программы в режиме игры

На рисунке 3.5 представлен интерфейс основного окна программы в режиме конструктора, в который можно перейти с помощью переключателя в верхнем правом углу. Для управления объектами на сцене используется панель справа.

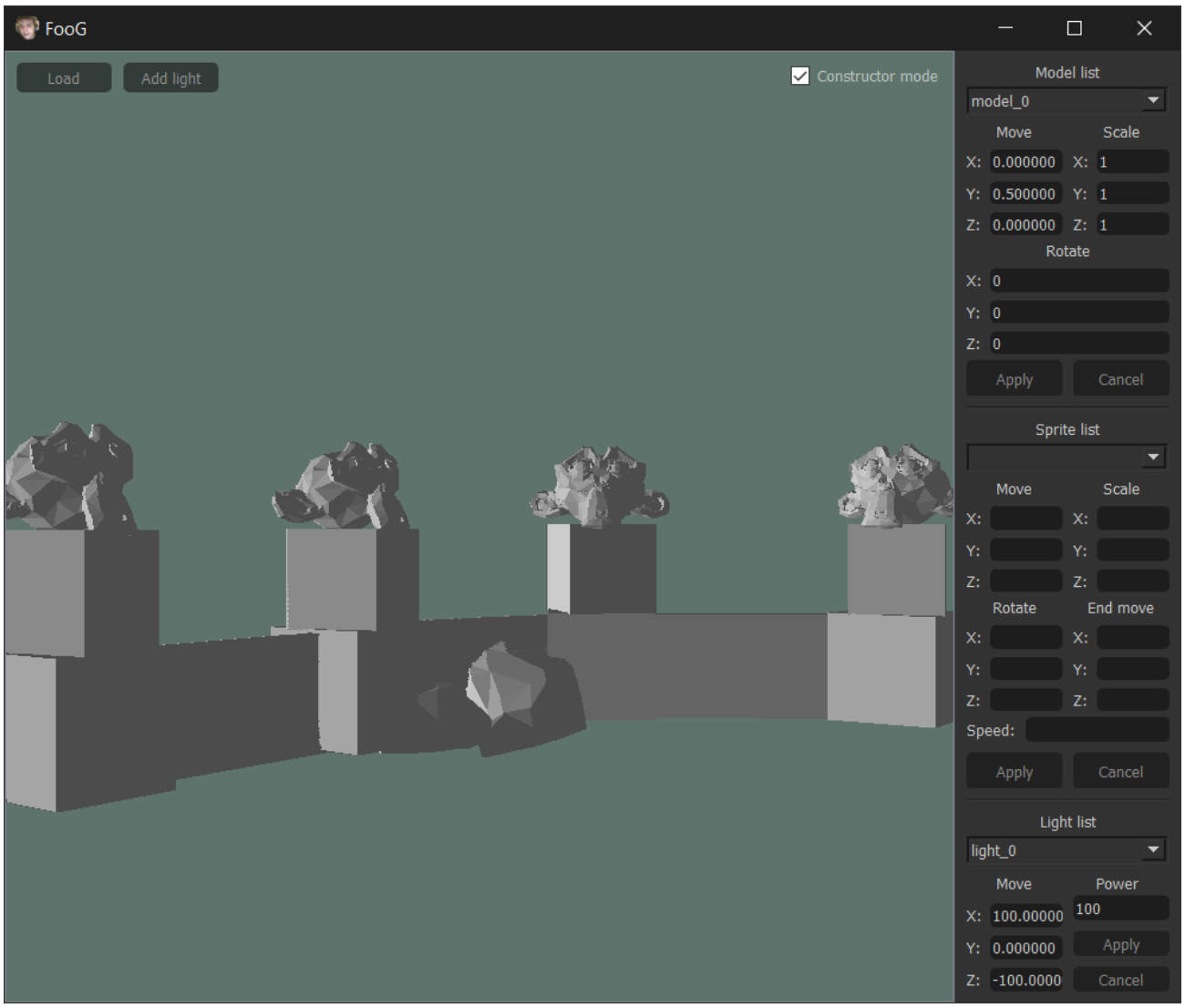

Рисунок 3.5 Интерфейс основного окна программы в режиме конструктора

Для взаимодействия с камерой используется набор клавиш wasd – для передвижения по сцене, ijkl – для изменения направления взгляда вверх/вниз и вправо/влево. Для демонстрации, приведу ту же сцену, но под другим углом, рисунок 3.6.

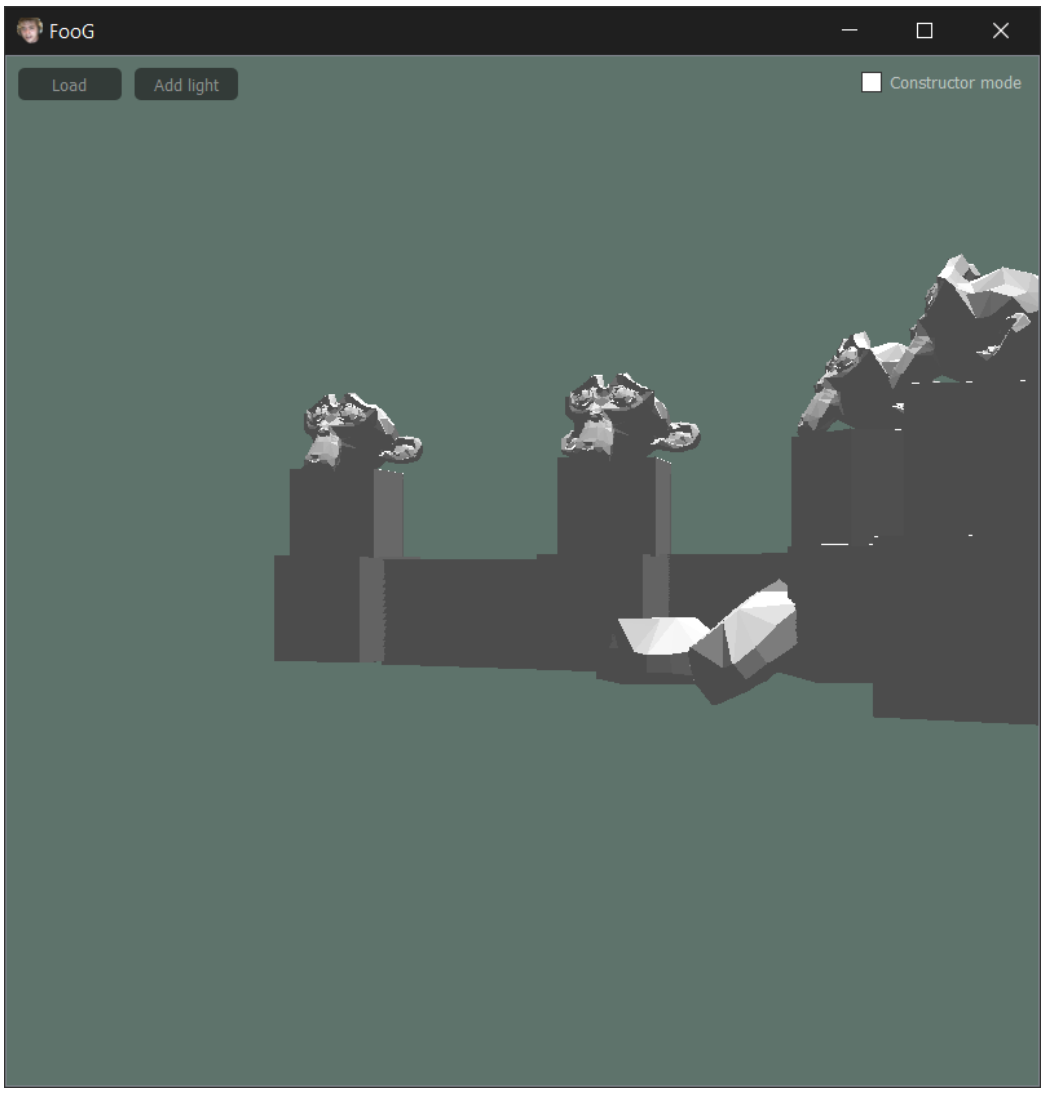

Рисунок 3.6 Демонстрация подвижной камеры

Для добавления моделей или загрузки уровня используется кнопка “Load” слева сверху, которая вызывает дополнительное меню, рисунок 3.7. В нём можно загрузить свою модель или уровень из .obj файла.

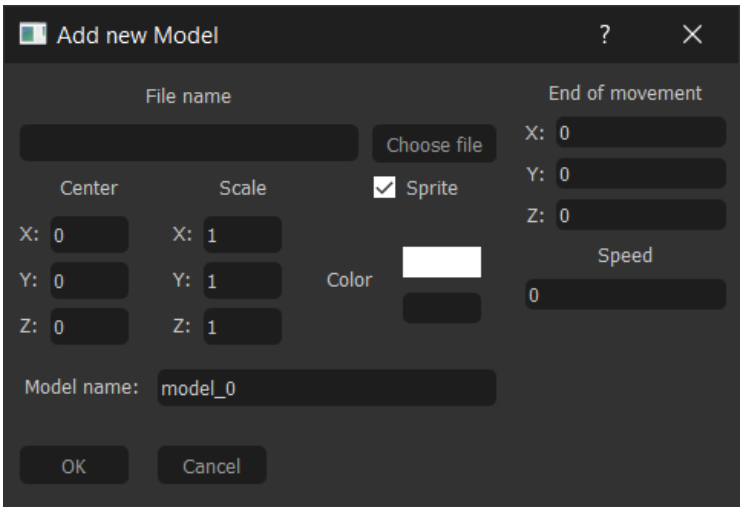

Рисунок 3.7 Интерфейс доп. окна для загрузки моделей и уровней

Для добавления источников света используется кнопка “Add Light” слева сверху, которая также вызывает дополнительное меню, рисунок 3.8.

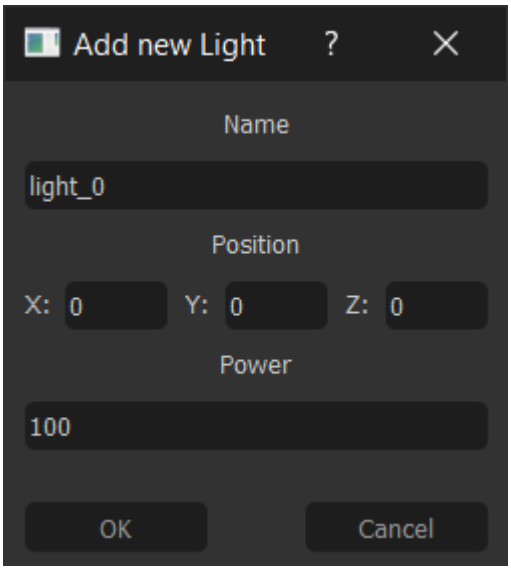

Рисунок 3.8 Интерфейс доп. окна для добавления источников света

В основном окне, в режиме конструктора, имеется блок моделей, где хранятся все добавленные модели и уровни. В блоке можно изменять положение, масштаб, а также повернуть выбранный объект, рисунок 3.9.

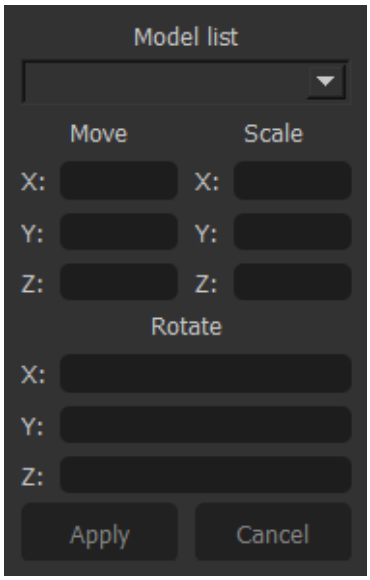

Рисунок 3.9 Блок редактирования моделей

Там же имеется блок спрайтов, для изменения позиции, масштаба и выполнения поворота. Также для спрайтов можно задать точку, в которую он должен переместится и его скорость, рисунок 3.10.

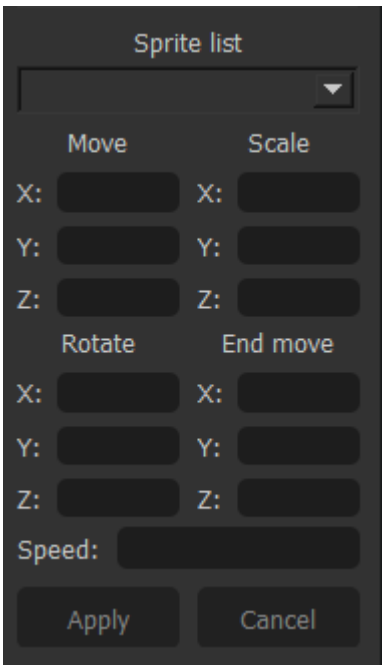

Рисунок 3.10 Блок редактирования спрайтов

Далее, находится блок редактирования источников света, где источник можно переместить и задать ему новое значение мощности, рисунок 3.11.

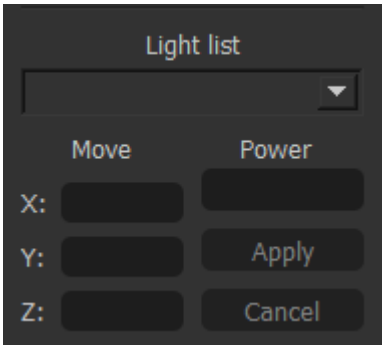

Рисунок 3.11 Блок редактирования источников света

Вывод

В данном разделе были выбраны средства реализации, описаны этапы реализации, а также рассмотрен интерфейс программы.

4. Исследовательская часть

В данном разделе будут проведены постановка эксперимента и сравнительный анализ алгоритмов на основе экспериментальных данных.

Для программы, которая изображает динамическую сцену, как отмечалось выше, важна скорость отрисовки сцены. Замедлить скорость отрисовки способны объекты с достаточно большим количеством вершин и граней. Также при просчёте освещения, требуется обойти каждую вершину из набора и подсчитать её интенсивность. Всё это в целом нагружает систему и снижает производительность.

4.1 Цель эксперимента

В рамках данной курсовой работы будет проведено исследование зависимости времени отрисовки от количества вершин объектов, которые находятся на сцене. Критерием измерения будет среднее время отрисовки кадра при собранной сцене.

4.2 План эксперимента

Измерения проводятся при одном источнике освещения, который для всех тестов находится в одной позиции, на сцену загружаются модели с количеством вершин: 88, 264, 473, 1258, 2516, 3774, 4528. Каждый замер производится 10 раз, за результат выбирается среднее арифметическое времени отрисовки кадра.

Так как оценивается не только скорость синтеза изображения, но и быстродействие обновления положения объектов относительно положения и направления взгляда камеры, то при просчёте камера будет передвигаться и поворачиваться, но так, что объекты не выходят за пределы рабочей области.

4.3 Результат эксперимента

По результатам измерений времени можно составить таблицу 4.1 и диаграмму 4.1.

Таблица 4.1 Результаты измерения времени отрисовки сцены

Количество вершин на сцене	88	264	473	1258	2516	3774	4528
Среднее время отрисовки в миллисекундах	39	44	50	59	77	101	114

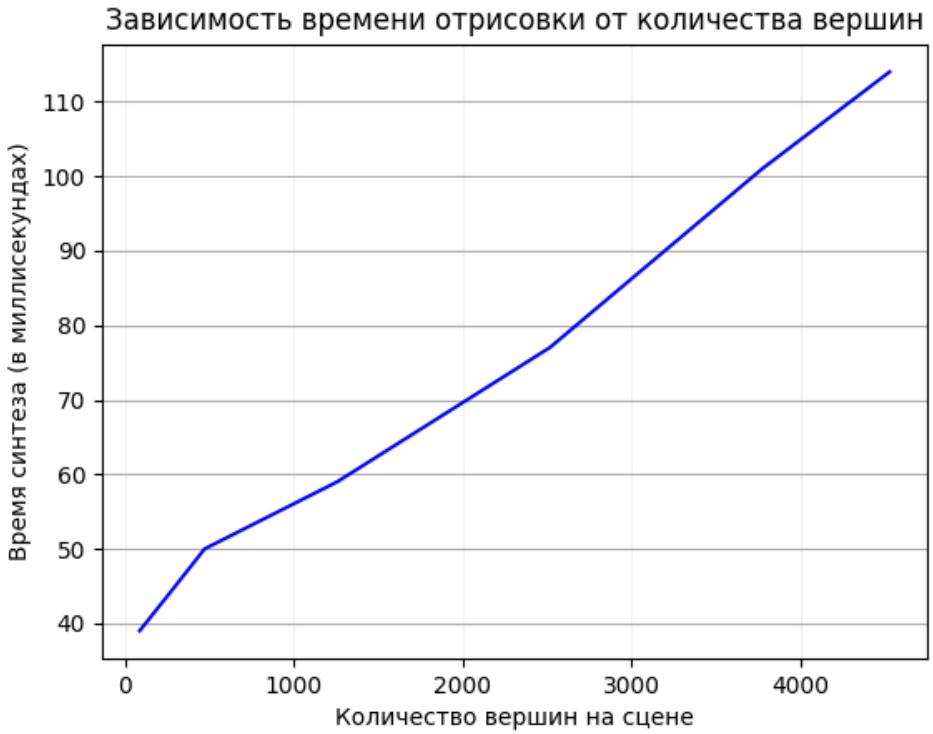

Диаграмма 4.1 Результаты измерения времени отрисовки сцены

Эксперименты проводились на компьютере с характеристиками:

- ОС – Windows 10, 64 бит;
- Процессор – Intel Core i5 7300HQ (2500 МГц, 4 ядра, 4 логических процессоров);
- Объём ОЗУ – 8 Гб.

Вывод

По результатам эксперимента можно заключить, что увеличение количества вершин значительно влияет на скорость отрисовки изображения. При использовании моделей с 2516 вершинами становятся заметны задержки между кадрами и снижается плавность при перемещении или поворотах камеры. При меньшем количестве вершин, эффект менее заметен.

Заключение

Цель курсовой работы достигнута. Спроектировано и реализовано программное обеспечение для визуализации трёхмерной сцены, которое наполнено статичными объектами – элементами декора, динамичными элементами – моделями в виде спрайтов и источниками света.

В ходе работы были проанализированы существующие алгоритмы удаления невидимых линий и поверхностей, модели освещения, закраски и указаны их преимущества и недостатки. Разработаны собственные и адаптированы существующие структуры данных и алгоритмы, необходимые для решения поставленной задачи.

Разработанный продукт является достаточно хорошим основанием для дальнейшей реализации конечной игры, а при выбранном способе добавления моделей и спрайтов на сцену, можно реализовать текстурирование и построение теней, что может значительно повысить качество выводимого изображения.

Список использованной литературы

1. Вишнякова Д. Ю., Надолинский Н. А. Программная реализация трехмерных сцен // Известия ЮФУ. Технические науки. 2001. №4.
2. Роджерс Д. Алгоритмические основы машинной графики. – М., «Мир», 1989
3. В. П. Иванов, А. С. Батраков. Трёхмерная компьютерная графика / Под ред. Г. М. Полищука. — М.: Радио и связь, 1995. — 224 с.
4. Дж. Ли, Б. Уэр. Трёхмерная графика и анимация. — 2-е изд. — М.: Вильямс, 2002. — 640 с.
5. Mark de Berg. Computational Geometry: Algorithms and Applications. — Springer Science & Business Media, 2008. — P. 259.