|  |  |
| --- | --- |
| **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\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Батраков Денис Вадимович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*фамилия, имя, отчество*

Группа \_\_\_\_ИУ7-42Б\_\_\_\_\_\_

Тип практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_стационарная\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_**Батраков Д.В.\_**\_\_\_\_**

*подпись, дата фамилия, и.о.*

Руководитель практики **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_**Куров А.В.**\_\_\_\_\_\_\_**

*подпись, дата фамилия, и.о.*

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

*Москва, 2019 г.*

**Индивидуальное задание:**

Разработать программу для визуализации тумана в трехмерной городской среде. Выбрать методы построения реалистичных изображений и визуализации тумана.

Оглавление

[Введение 4](#_Toc14169990)

[1. Аналитическая часть 5](#_Toc14169991)

[1.1 Формализация объектов синтезируемой сцены 5](#_Toc14169992)

[1.2 Анализ алгоритмов удаления невидимых линий и поверхностей 6](#_Toc14169993)

[1.3 Анализ методов закрашивания 9](#_Toc14169994)

[1.4 Анализ алгоритмов построения теней 11](#_Toc14169995)

[2. Конструкторская часть 13](#_Toc14169997)

[2.1 Общий алгоритм решения задачи 13](#_Toc14169998)

[2.2 Алгоритм Z-буфера 13](#_Toc14169999)

[2.3 Простой метод освещения 14](#_Toc14170000)

[2.4 Выбор используемых типов и структур данных 14](#_Toc14170002)

[3. Технологическая часть 15](#_Toc14170003)

[3.1 Выбор и обоснование языка программирования и среды разработки 15](#_Toc14170004)

[3.2 Диаграмма связей классов(UML) 15](#_Toc14170005)

[Заключение 18](#_Toc14170008)

[Список использованной литературы 19](#_Toc14170009)

[Приложения 20](#_Toc14170010)

# Введение

В наше время сложно представить компьютерную программу, которая бы не использовала компьютерную графику, т.к. она используется везде для наглядного отображения данных в кино, играх и многом другом.

Вследствие этого перед разработчиками ставится задача создания реалистичных изображений, которые бы учитывали оптические явления (преломление, отражение, рассеивание света), текстуру, цвет. А для еще более реалистичных изображений нужно учитывать дифракцию, вторичные отражения и интерференцию света.

Для решения такого большого спектра задач были разработаны алгоритмы компьютерной графики. Главная их проблема – это ресурсозатратность. Т.к. чем более реалистичное изображение мы хотим получить, тем больше времени и памяти нам понадобится для его синтеза. Данный факт становится проблемой при создании динамической сцены.  
  
Моей целью во время летней практики будет модифицирование или выбор существующих алгоритмов компьютерной графики для создания трехмерной сцены, а также составление uml схемы программы.

# 1. Аналитическая часть

## 1.1 Формализация объектов синтезируемой сцены

Сцена состоит из:

* Камер – представляют собой материальную точку, из которой наблюдатель смотрит на сцену

Положение задается трехмерными координатами и вектором взгляда. Будем считать, что наблюдатель находится в бесконечности, для облегчения задачи синтеза изображений.

* Источников света –материальная точка, из которой во все стороны исходят лучи света. В частном случае, если источник света расположен в бесконечности, имеет направленность.

Положение задается трехмерными координатами.

Цвет свечения описывается через RGB или иные параметры.

* Солнца/луны – источник света, который движется по полуокружности и меняет свой цвет в зависимости от угла наклона. Являются источниками света.
* Плоскость земли – ограничивающая плоскость. Предполагается, что под ней не расположено объектов. Изначально расположена внизу экрана (на максимальной координате y) и параллельна oXZ.

Размеры задаются шириной, длиной и цветом (RGB или иной системой).

* Здания – правильная призма, основания которых параллельны плоскости земли. Боковые ребра перпендикулярны плоскости земли.

Задается

Координатами положения центра основания на плоскости земли (x, z)

Высотой h

Количеством боковых граней(n).

* Крыша здания – это часть здания. Типы: плоская (без крыши), пирамидальная (основание которой совпадает с верхним основанием здания).

Пирамидальная крыша задается:

Высотой

Положением центра основания (x, y, z)

Количеством боковых граней.

* Туман – прозрачность, накладывающаяся на каждый пиксель сцены. Ее степень зависит от того как далеко располагается видимый пиксель на сцене.

Здания и крыши наилучшим образом можно описать через поверхностные модели, потому что каркасные модели не дадут реалистичное изображение, которого мы стараемся достичь, а объемная модель будет более затратной по памяти.

Поверхностную модель можно задается несколькими способами:

**Полигональной сеткой** – совокупность вершин, ребер и граней, которые определяют форму объекта (Рис.1).

* Вершинное представление: вершины хранят указатель на другие вершины, с которыми они связаны. Для создания списка граней, который будет использоваться при рендеринге, нужно обойти все данные(вершины), что затрудняет работу с ним.
* Список граней представляет собой объект как множество граней и вершин.
* Таблица углов - хранит вершины в заранее определенной таблице. Не подойдет для моей задачи, так как изменение данного представления весьма затратное по времени.

**Параметрическим представлением** – для того, чтобы получить поверхности нужно вычислять функцию, зависящую от параметра. Из-за того, что на сцене нет никаких поверхностей вращения, использование параметрического представления будет затруднительным.

Исходя из информации выше, наиболее удобным способом для хранения сцены при решении моей задачи является список граней из-за того, что данные в нем можно эффективно преобразовывать. Данное представление позволяет осуществлять явный поиск граней, окружающих вершину и вершин грани.

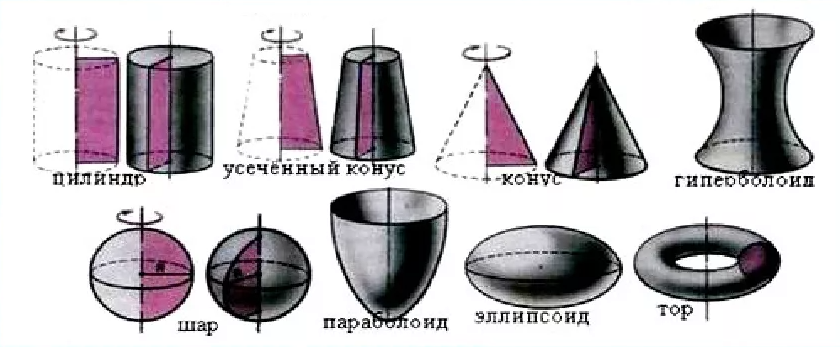


Рисунок 1 Примеры тел вращения

## 1.2 Анализ алгоритмов удаления невидимых линий и поверхностей

При выборе алгоритма удаления невидимых линий нужно учитывать особенности поставленной задачи. Речь идет о том, что будет несколько камер, которые можно будет двигать, из-за чего придется часто менять изображение. Ввиду данного нюанса алгоритм, который будет использоваться, должен работать быстро, иначе при смене позиции камеры или вектора ее направления пользователю придется ждать рендеринг сцены, что сильно скажется на положительном опыте использования приложения. Надо выбрать такой алгоритм, который позволит добиться максимальной производительности даже на слабых компьютерах.

**Алгоритм обратной трассировки лучей**

Алгоритм очень универсален, из-за чего он работает не так быстро, как хотелось бы. Обратная трассировка позволяет работать с несколькими источниками света и передавать множество разных оптических явлений, что для моей задачи не сильно нужно.

Учитывая, что туман состоит из мельчайших капелек воды, то при рендеринге реалистичного тумана, согласно алгоритму, нужно было бы следуя правилам обратной трассировки, каждую каплю рассматривать как отдельный объект сцены, внутри которого могут возникать такие явления как дисперсия, внутреннее отражение и преломление света. Данный способ очень сильно расходует системные ресурсы, но позволяет моделировать явления радуги, что не входит в задачи моей программы.

Положительные стороны:

1. Возможность использования в параллельных вычислительных системах (т.к. расчет отдельной точки выполняется независимо от других точек).

Отрицательные стороны:

1. Большое количество вычислений для синтеза изображения сцены.

**Вывод:** так как на моей сцене не будут подчеркиваться явления преломления и отражения света, то использование алгоритмов трассировки будет излишним. При заметном замедлении работы программы, качество изображения заметно не улучшится.

**Алгоритм, использующий Z буфер**

Одним из главных плюсов данного алгоритма является простота его реализации, которая не сказывается негативно на задаче удаления поверхностей и визуализации их пересечения. Т.к. в этом алгоритме не тратится время на сортировку элементов сцены, мы получаем преимущество в скорости работы. Особенно полезным это становится при большом количестве домов в сцене.

Так как размер синтезируемого изображения достаточно мал, то затраты по памяти, при хранении информации о каждом пикселе в данном алгоритме очень малы для современных компьютеров.

**Алгоритм Робертса**

Положительной стороны:

1. Точность вычислений. Она достигается за счет работы в объектном пространстве, в отличии от большинства других алгоритмов.

Отрицательные стороны:

1. Вычислительная трудоемкость. В теории она растет как квадрат количества объектов(O(n^2)). Поэтому, при большом количестве домов на сцене, этот алгоритм будет показывать себя как недостаточно быстрый. Можно использовать разные оптимизации для повышения эффективности, например, сортировку по z.
2. Некоторые из оптимизаций крайне сложны, что затрудняет реализацию этого алгоритма.

**Алгоритм Варнока**

Алгоритм основывается на рекурсивном разбиении экрана. В зависимости от расположения объектов это может стать, как положительной, так и отрицательной стороной алгоритма. Чем меньше пересечений объектов, тем быстрее алгоритм завершит свою работу.

**Вывод**

Алгоритм метода трассировки лучей не позволит в реальном времени смоделировать туман из-за рассеивания света. Можно лишь имитировать его используя зависимость затуманенности от пройденного расстояния луча.

Поэтому предпочтительнее использовать Z буфер, на его основе будет просто визуализировать туман. В случае тумана можно хранить глубину пикселя, с помощью которой можно будет вычислить насколько «затуманенным» он будет.

## 1.3 Анализ методов закрашивания

**Простая закраска**

Вся грань объекта закрашивается одним уровнем интенсивности, который высчитываемый по закону Ламберта.

Данный метод весьма прост в реализации и не требователен к ресурсам. Однако плохо подходит для тел вращения (Рис.2), т.к. плохо учитывает отраженный свет.

Для решения моей задачи этот метод очень хорошо подходит, так как вся работа ведется с гранями зданий и тел вращения нет.

**Закраска по Гуро**

Этот метод использует билинейную интерполяцию интенсивностей, благодаря которой устраняется дискретность изменения интенсивности и создается иллюзия гладкой криволинейной поверхности. Хорошо сочетается с диффузным отражением.

**Закраска по Фонгу**

Этот метод использует линейную интерполяцию векторов нормалей. Благодаря этому достигается лучшая локальная аппроксимация кривизны поверхности. Изображение выходит более реалистичным, зеркальные блики выглядят правдоподобнее, чем в методе закраски по Гуро.

Однако закраска Гуро требует меньших вычислительных затрат чем закраска по Фонгу, так как интерполируются значения векторов нормалей, на основе которых потом вычисляется интенсивность.

**Вывод:** так как фигуры сцены состоят из плоскостей, закраска по Фонгу и Гуро будет скорее мешать: ребра зданий будут сглажены. Тело здания будет хуже восприниматься (Рис.2). Поэтому лучше всего использовать простую закраску.

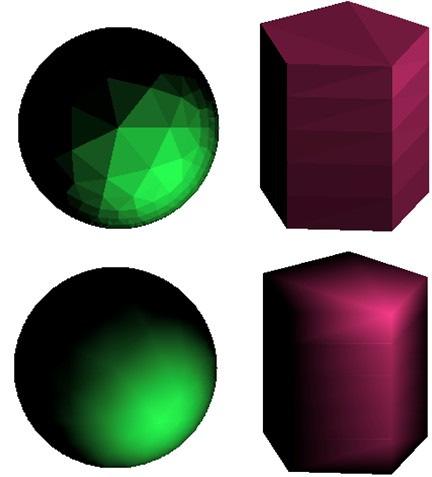


Рисунок 2 Сравнение методов закрашивания

## 1.4 Анализ алгоритмов построения теней

При трассировке лучей тени получаются без дополнительных вычислений.  
Пиксель затенен, когда луч попадает на объект и позже не попадает ни в объект, ни в источник света.

Так как в анализе алгоритмов трассировка лучей не была выбрана в качестве алгоритма синтеза сцены, то тени нужно вычислять отдельно.

Один из способов нахождения теней – вычисление проекций тел.

Можно использовать метод теневых карт, в котором предполагается, что освещены только те фрагменты, которые видны из положения источника. Находить видимость можно с помощью алгоритма Робертса, алгоритма Z буфера и других.

Для создания теневых карт будет использоваться алгоритм Z буфера так как этот алгоритм позволит быстро найти видимость объектов сцены.

# 2. Конструкторская часть

## 2.1 Общий алгоритм решения задачи

1. Задать объекты сцены

2. Задать источники света (учет перемещения солнца) и положение наблюдателя

3. Для каждого полигона высчитать нормаль и интенсивность цвета, найти внутренние пиксели

4. Найти тени

5. Используя алгоритм Z буфера получить изображение сцены, сохранить Z буфер для дальнейших расчетов

6. Отобразить изображение.

## 2.2 Алгоритм Z-буфера

1. Всем элементам буфера кадра присвоить фоновое значение
2. Инициализировать Z буфер минимальными значениями глубины
3. Выполнить растровую развертку каждого многоугольника сцены:
   1. Для каждого пикселя, связанного с многоугольником вычислить его глубину z(x, y)
   2. Сравнить глубину пикселя со значением, хранимым в Z буфере.

Если z(x, y) > zбуф(x, y), то zбуф(x,y) =z(x,y), цвет(x, y) = цвет пикселя.

1. Отобразить результат

## 2.3 Простой метод освещения

В простом методе освещения интенсивность рассчитывается по закону Ламберта:

I = I0\*cos(α), где

I – результирующая интенсивность света в точке

I0 – интенсивность источника

α – угол между нормалью к поверхности и вектором направления света

## 2.4 Выбор используемых типов и структур данных

* Источник света – задается расположением и направленностью света.
* Объекты сцены – задаются вершинами и гранями.
* Математические абстракции
  + Точка – хранит координаты x, y, z
  + Вектор – хранит направление по x, y, z
  + Многоугольник – хранит вершины, нормаль, цвет
* Интерфейс – используются библиотечные классы для предоставления доступа к интерфейсу.

# 3. Технологическая часть

## 3.1 Выбор и обоснование языка программирования и среды разработки

В качестве языка программирования был выбран C++ т.к.:

* Я ознакомилась с этим языком программирования во время занятий по компьютерной графике, что сократит время написания программы
* Данный язык программирования объектно-ориентирован, что даст в полной мере
  + использовать наследование, абстрактные классы и т.д.
  + представлять трехмерные объекты сцены в виде объектов классов, что позволит легко организовать взаимодействие между ними, положительно влияя на читабельность, не снижая эффективности.

В качестве среды разработки была выбрана «CLion» т.к.:

* Она бесплатна в пользовании студентами;
* Имеет множество удобств, которые облегчают процесс написания и отладки кода;
* Обеспечивает работу с Qt5 – графической библиотекой для отображения интерфейса и взаимодействия с логикой его частей
* я знаком с данной средой разработки, что сократит время изучения возможностей среды.

## 3.2 Диаграмма связей классов(UML)

Посмотреть на uml можно в разделе приложения(Рис 3-11). При ее составлении были использованы паттерны проектирования позволяющие сделать код обслуживаемым и модернезируемым.

# Заключение

Во время выполнения поставленной задачи были рассмотрены и проанализированы основные алгоритмы удаления невидимых линий, построения теней, методы закрашивания. 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 Qt5, 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](https://ru.wikipedia.org/wiki/Полигональная_сетка) (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.

# Приложение

Рисунок 3 Реализация связей объектов сцены

# 

# 

Рисунок 4 Реализация поверхностной модели

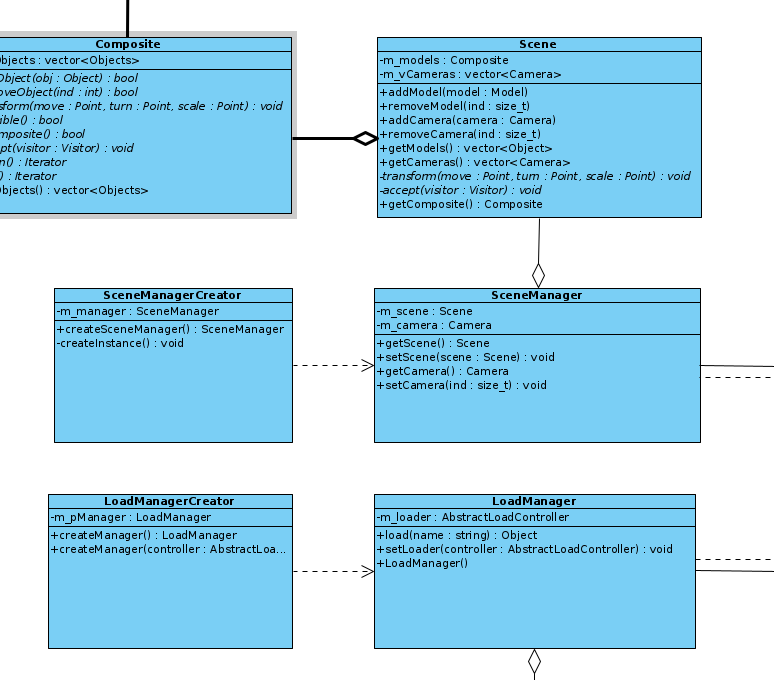


Рисунок 5 Реализация сцены

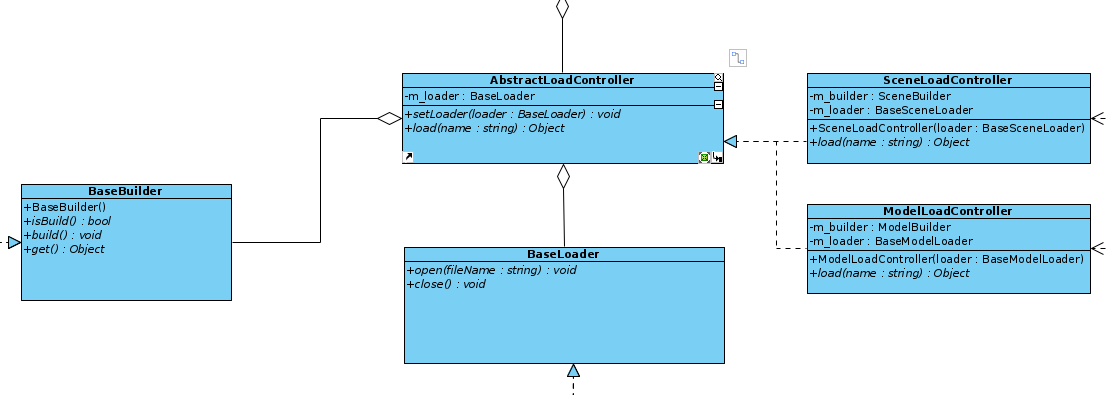


Рисунок 5 Реализация контроллера сцены и моделей

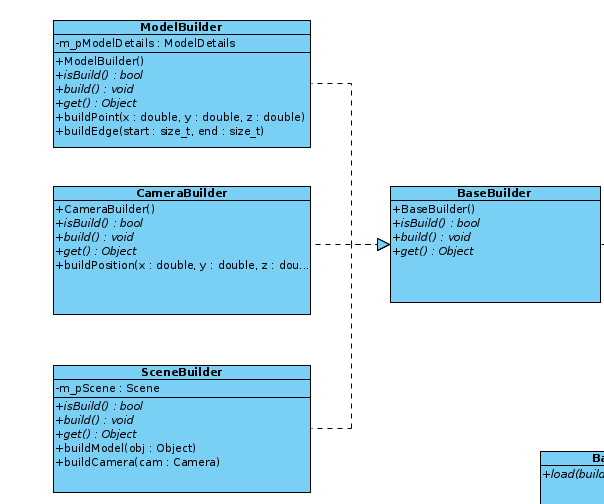


Рисунок 6 Реализация строителей объектов и сцены

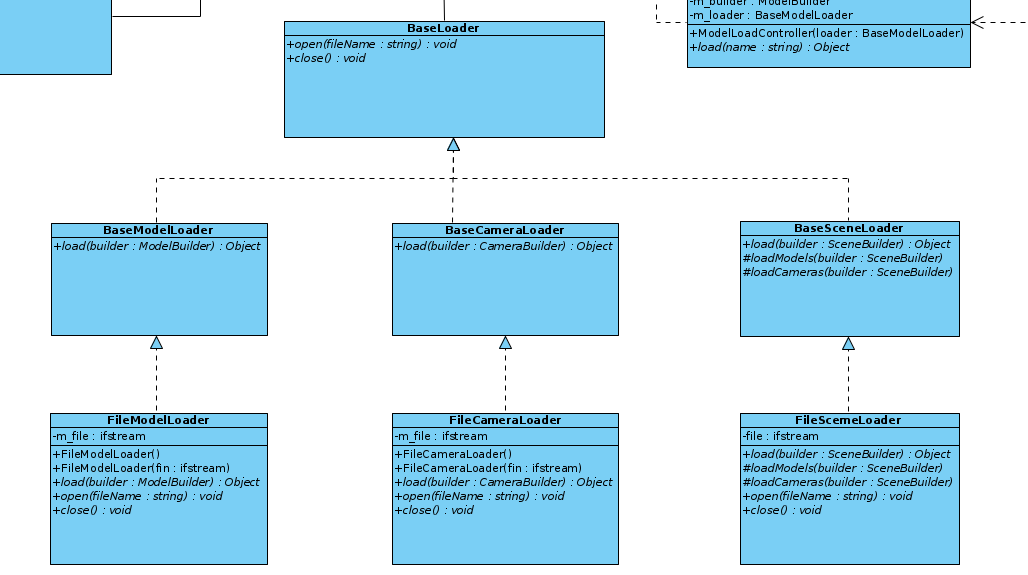


Рисунок 7 Реализация загрузки данных

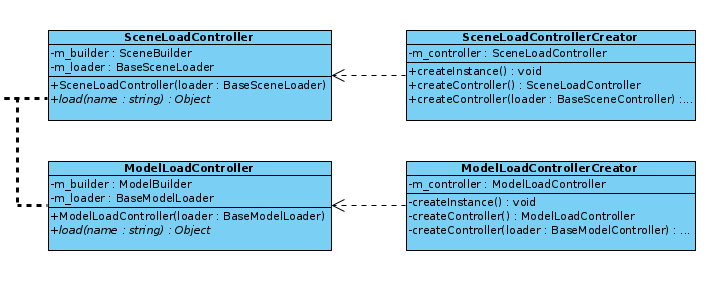


Рисунок 8 Реализация контроллеров сцены и модели

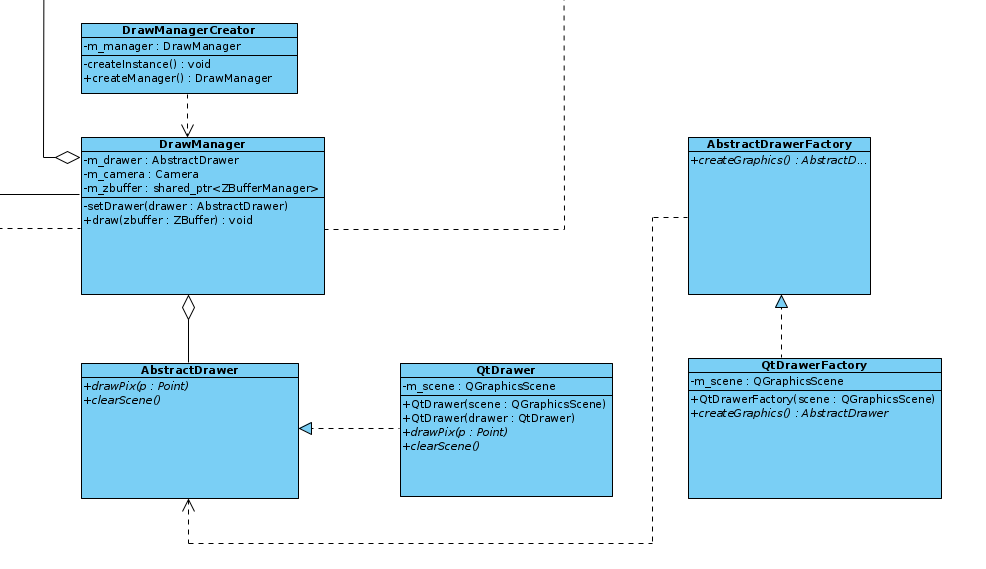
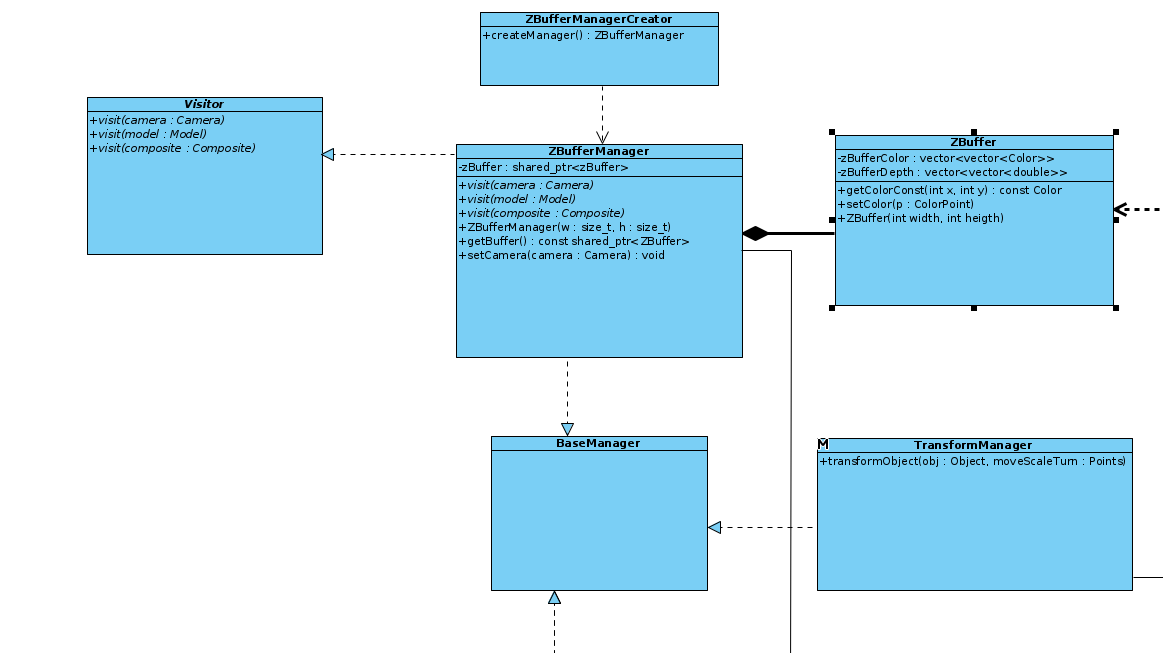


Рисунок 9 Реализация отрисовки сцены

Рисунок 10 Реализация z-буффера

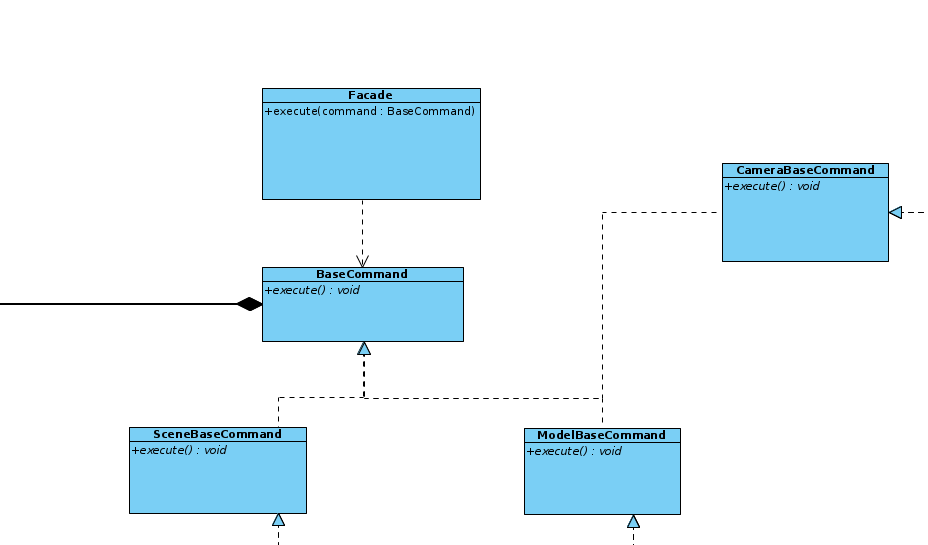


Рисунок 11 Реализация команд и высокоуровненого фасада