Государственное образовательное учреждение высшего профессионального образования

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана»***  ***(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:**

Разработка программы построения изображения комнаты.

Студент  **\_\_\_\_\_\_\_\_\_\_\_\_**  Бабарыкин Д.С.

(Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта \_\_\_\_\_\_\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_ \_**Ломовской И.В.

(Подпись, дата) (И.О.Фамилия)

Оглавление

[Введение 4](#_Toc406758009)

[1 Аналитический раздел 5](#_Toc406758010)

[1.1 Распространённые алгоритмы удаления невидимых линий и поверхностей 5](#_Toc406758011)

[1.1.1 Алгоритм Робертса 5](#_Toc406758012)

[1.1.2 Алгоритм трассировки лучей 6](#_Toc406758013)

[1.1.3 Алгоритм Варнока 6](#_Toc406758014)

[1.1.4 Алгоритм, использующий z-буфер 7](#_Toc406758015)

[1.2 Распространённые алгоритмы закрасок 8](#_Toc406758016)

[1.2.1 Модель освещения Ламберта 8](#_Toc406758017)

[1.2.2 Затенение по Гуро 9](#_Toc406758018)

[1.2.3 Закраска по Фонгу 9](#_Toc406758019)

[1.3 Выводы и выбор алгоритмов для поставленной задачи 10](#_Toc406758020)

[2 Конструкторский раздел 11](#_Toc406758021)

[2.1 Структуры данных 11](#_Toc406758022)

[2.2 Общий алгоритм построения изображения в программе 12](#_Toc406758023)

[2.2 Поворот сцены 14](#_Toc406758024)

[2.3 Сдвиг сцены 14](#_Toc406758025)

[2.4 Перспективные преобразования 15](#_Toc406758026)

[2.5 Фильтрация поверхностей 15](#_Toc406758027)

[2.5.1 Понятие фильтрации поверхностей 15](#_Toc406758028)

[2.5.2 Фильтрация нелицевых граней 16](#_Toc406758029)

[2.5.3 Фильтрация заведомо полностью загороженных граней 17](#_Toc406758030)

[2.6 Совмещение алгоритма, использующего z-буфер, и закраски Гуро 18](#_Toc406758031)

[2.7 Распараллеливание 19](#_Toc406758032)

[3 Технологический раздел 21](#_Toc406758033)

[3.1 Выбор языка программирования 21](#_Toc406758034)

[3.2 Выбор формата файлов описания объектов сцены 21](#_Toc406758035)

[3.3 Архитектура программы 21](#_Toc406758036)

[3.3.1 Общая структура и формы 21](#_Toc406758037)

[3.3.2 Общий порядок работы с программой 22](#_Toc406758038)

[3.3.3 Структуры данных 22](#_Toc406758039)

[3.3.4 Структура и паттерны проектирования 24](#_Toc406758040)

[3.3.5 Паттерн Команда (Command) 25](#_Toc406758041)

[3.3.6 Паттерн Одиночка (Singleton) 26](#_Toc406758042)

[3.3.6 Паттерн Фасад (Facade) 26](#_Toc406758043)

[3.3.7 Паттерн Стратегия 27](#_Toc406758044)

[3.4 Интерфейс программы 28](#_Toc406758045)

[3.4.1 Форма введения размеров сцены 28](#_Toc406758046)

[3.4.2 Форма редактора интерьера 29](#_Toc406758047)

[3.4.3 Форма обозревателя комнаты 30](#_Toc406758048)

[4 Исследовательская часть 31](#_Toc406758049)

[4.1 Фильтрация уже полностью загороженных граней 31](#_Toc406758050)

[4.2 Временн*а*я оценка увеличения производительности за счёт фильтраций 31](#_Toc406758051)

[Заключение 33](#_Toc406758052)

[Список литературы 34](#_Toc406758053)

# Введение

В наши дни компьютеры используются во многих сферах деятельности. Самая важная функция компьютера – обработка информации. Особо можно выделить обработку информации, связанную с изображениями. Компьютерная графика – это область деятельности, в которой компьютеры используются для синтеза изображений и для обработки визуальной информации. В настоящее время 3D-моделирование является важнейшей областью компьютерной графики, так как построение трехмерного изображения, близкого к реальному, является необходимой составляющей в большой доле программных продуктов.

Цель моего курсового проекта – разработать программу построения изображения комнаты. Программа должна содержать:

1. Схематический редактор сцены
2. Окно отображения комнаты

Редактор должен позволять задавать конфигурацию комнаты: размеры помещения, расстановку предметов и их количество. Необходима возможность сохранения получившейся сцены в файл и загрузка уже готовой из файла. Изменяемым параметром для окна отображения комнаты является положение наблюдателя. То есть сцену нужно уметь вращать, приближать и отдалять.

# 1 Аналитический раздел

## 1.1 Распространённые алгоритмы удаления невидимых линий и поверхностей

Для того чтобы выбрать подходящий алгоритм построения изображения, необходимо осуществить краткий обзор известных алгоритмов и осуществить выбор наиболее подходящего для решения поставленной задачи.

### 1.1.1 Алгоритм Робертса

Алгоритм Робертса представляет собой первое известное решение задачи об удалении невидимых линий. Это метод, работающий в объектном пространстве. В соответствии с алгоритмом, прежде всего 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.

Преимущества данного алгоритма в том, что математические методы, используемые в нем, просты, мощны и точны. Более поздние реализации алгоритма, например использующие предварительную сортировку вдоль оси z, демонстрируют почти линейную зависимость от числа объектов.

Минус этого алгоритма в том, что вычислительная трудоемкость алгоритма Робертса растет теоретически, как квадрат числа объектов. Реализация оптимизированных алгоритмов весьма сложна.

### 1.1.2 Алгоритм трассировки лучей

В этом методе для каждого пикселя картинной плоскости определяется ближайшая к нему грань, для чего через этот пиксель выпускается луч, находятся все его пересечения с гранями и среди них выбирается ближайшая.

К достоинствам данного алгоритма можно отнести возможность получения изображения гладких объектов без аппроксимации их примитивами (например, треугольниками). Вычислительная сложность метода линейно зависит от сложности сцены. Нетрудно реализовать наложение света и тени на объекты. Качество полученного изображения получается очень реалистичным, этот метод отлично подходит для создания фотореалистичных картин.

Серьёзным недостатком алгоритма трассирования является производительность. Для получения изображения необходимо создавать огромное число лучей, проходящих через сцену и отражаемых от объекта. Это приводит к существенному снижению скорости работы программы.

### 1.1.3 Алгоритм Варнока

Алгоритм Варнока работает в пространстве изображений. В основу алгоритма положен принцип "разделяй и властвуй", состоящий в разбиении области рисунка на более мелкие подобласти (окна). Для каждой подобласти (окна) определяются связанные с ней многоугольники и те из них, видимость которых определить "легко", изображаются на экране. В противном же случае разбиение повторяется, и для каждой из вновь полученных подобластей рекурсивно применяется процедура принятия решения. Предполагается, что с уменьшением размеров области ее перекрывает все меньшее и меньшее количество многоугольников. Считается, что в пределе будут получены области, содержащие не более одного многоугольника, и решение будет принято достаточно просто. Если же в процессе разбиения будут оставаться области, содержащие не один многоугольник, то следует продолжать процесс разбиения до тех пор, пока размер области не станет совпадать с одним пикселом. В этом случае для полученного пикселя необходимо вычислить глубину (значение координаты Z) каждого многоугольника и визуализировать тот из них, у которого максимальное значение этой координаты.

### 1.1.4 Алгоритм, использующий z-буфер

Данный алгоритм удаления невидимых поверхностей является одним из простейших. Этот алгоритм работает в пространстве изображения. Здесь обобщается идея о буфере кадра. Буфер кадра используется для заполнения атрибутов (интенсивности) каждого пикселя в пространстве изображения. Наряду с буфером кадра вводится Z-буфер, представляющий собой специальный буфер глубины, в котором запоминаются координаты Z (глубина) каждого видимого пикселя в пространстве изображения. В процессе работы глубина (значение координаты Z) каждого нового пикселя, который надо занести 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 пиксел, уже находящийся в буфере кадра, то новый пиксель заносится в буфер кадра. Помимо этого производится корректировка Z-буфера: в него заносится глубина нового пикселя. Если же глубина (значение координаты Z) нового пикселя меньше, чем хранящегося в буфере, то никаких действий производить не надо. В сущности, алгоритм для каждой точки (x,y) находит наибольшее значение функции Z(x,y).

Этот алгоритм несмотря на свою простоту позволяет удалять сложные поверхности и позволяет визуализировать пересечения таких поверхностей. Сцены могут быть произвольной сложности, а поскольку размеры изображения ограничены размером экрана дисплея, то трудоемкость алгоритма имеет линейную зависимость от числа рассматриваемых поверхностей. Элементы сцены заносятся в буфер кадра в произвольном порядке, поэтому в данном алгоритме не тратится время на выполнение сортировок, необходимых в других алгоритмах.

## 1.2 Распространённые алгоритмы закрасок

### 1.2.1 Модель освещения Ламберта

Одной из самых простых моделей освещения является модель Ламберта. Она учитывает только идеальное диффузное отражение света от тела. Считается, что свет падающий в точку, одинакового рассеивается по всем направлением полупространства (см Рисунок 1). Таким образом, освещенность в точке определяется только плотностью света в точке поверхности, а она линейно зависит от косинуса угла падения. При этом положение наблюдателя не имеет значение, т.к. диффузно отраженный свет рассеивается равномерно по всем направлениям.

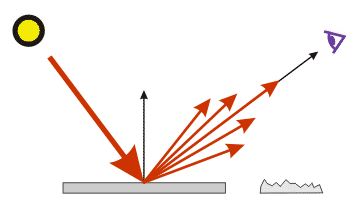


Рисунок 1

Пусть

α – угол между вектором направления света и нормалью к поверхности

I – результирующая интенсивность света в точке

I0 – интенсивность источника

k – способность материала к диффузному отражению

Тогда формула расчёта интенсивности будет иметь следующий вид:

I = I0\*k\*cos(α) (1)

Большим недостатком данной модели является то, что согласно приведённой выше формуле, все точки грани будут иметь одинаковую интенсивность.

### 1.2.2 Затенение по Гуро

Метод Гуро является одним из способов устранения дискретности интенсивностей закрашивания.

Данный алгоритм предполагает следующие шаги:

1. Вычисление векторов нормалей к каждой грани.
2. Вычисление векторов нормали к каждой вершине грани путем усреднения нормалей к граням (см. рисунок 2)
3. Вычисление интенсивности в вершинах грани.
4. Интерполяция интенсивности вдоль ребер грани.
5. Линейная интерполяция интенсивности вдоль сканирующей строки.

Закраска по Гуро хорошо сочетается с диффузным отражением. Данный метод интерполяции обеспечивает лишь непрерывность значений интенсивности вдоль границ многоугольников, но не обеспечивает непрерывность изменения интенсивности, следовательно, возможно появление полос Маха.

На рисунке 2 Nv – нормаль к вершине – усреднённое значение нормалей N1…NN.

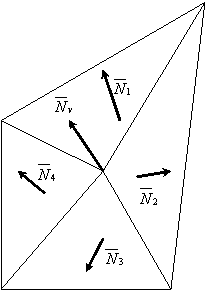


Рисунок 2

### 1.2.3 Закраска по Фонгу

Закраска по Фонгу требует б*о*льших вычислительных затрат, чем Гуро. При такой закраске, в отличие от метода Гуро, вдоль сканирующей строки интерполируется значение вектора нормали, а не интенсивности. Используя закраску по Фонгу, можно достичь лучшей локальной аппроксимации кривизны поверхности.

Изображение получается более реалистичным. Закраска по Фонгу хорошо сочетается с зеркальным отражением.

Шаги алгоритма:

1. Вычисление векторов нормалей в каждой грани.
2. Вычисление векторов нормали к каждой вершине грани.
3. Интерполяция векторов нормалей вдоль ребер грани.
4. Линейная интерполяция векторов нормалей вдоль сканирующей строки.
5. Вычисление интенсивности в очередной точке сканирующей строки.

Основным главными недостатками данного алгоритма являются ресурсоемкость и большая вычислительная сложность.

## 1.3 Выводы и выбор алгоритмов для поставленной задачи

В программе предвидится большое количество объектов на сцене, каждый из которых будет иметь множество граней. Также для создания удобной работы требуется высокая скорость отрисовки, близкая к реальному времени, поэтому необходим быстродействующий алгоритм, способный работать с большим количеством граней. Под эту роль хорошо подходит алгоритм, использующий буфер глубины. Для создания изображения, близкого к реалистичному необходимо использование одной из моделей освещения. Так как для удаления невидимых поверхностей мы выбрали Z-буфер, в котором используется интерполяция для вычисления глубины каждый точки, нам хорошо подойдёт закраска по Гуро, так же требующая интерполяцию. Благодаря возможности объединить некоторые интерполяционные вычисления и мы получаем возможность добиться хорошей скорости при достойном качестве изображения.

# 2 Конструкторский раздел

# 2.1 Структуры данных

Чтобы формализовать общий алгоритм синтеза изображения в данной программе, необходимо ввести определения использующихся в ней структур данных. Итак, будем считать, что

1) Сцена представляет собой список с произвольным числом моделей

2) Модель включает следующие данные:

* Массив вершин фигуры
* Массив векторов нормалей к вершинам
* Массив поверхностей
* Массив материалов

1. Поверхность содержит:

* Массив индексов, входящих в неё вершин, из массива вершин
* Массив индексов нормалей, входящих в неё вершин, из массива нормалей
* Индекс материала из массива материалов

1. Вершина и нормаль представляют собой три вещественных числа
2. Материал несёт информацию о способностях поверхности к диффузному отражению. В частности, это три вещественных числа в пределах от 0 до 1, каждое отвечающее за интенсивность в одном из трёх каналов цветов: красного, зелёного и синего. Таким образом, здесь используется широко распространённая аддитивная цветовая модель RGB (аббревиатура от red, green, blue).
3. Камера содержит:

* Сдвиги по осям X, Y, Z – 3 вещественных числа
* Углы поворота вокруг осей X, Y, Z – 3 вещественных числа

## 2.2 Общий алгоритм построения изображения в программе

Вход: сцена и камера

Выход: изображение (заполненный буфер кадра)

Алгоритм представлен на рисунке 3.

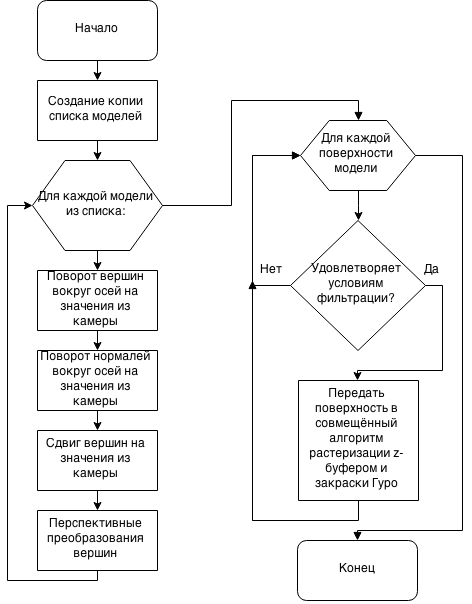


Рисунок 3

## 2.2 Поворот сцены

Чтобы создать у пользователя впечатление вращения камеры, мы будем последовательно поворачивать каждую фигуру сцены вокруг осей на значения, заданные в камере.

Поворот фигуры, как показано на схеме из рисунка 3, состоит из поворота каждой вершины фигуры и каждой нормали к вершине.

Для задачи просмотра комнаты уместно использовать повороты только вокруг осей X и Y. Для этого используются матрицы поворота. Если разложить результирующий вектор (x, y, z) покомпонентно, то получим формулы:

Вращение точки (x0, y0, z0) вокруг OX на угол α:

x = x0  (2)

y = y0 \* cos(α) – z0 \* sin(α)

z = y0 \* sin(α) + z0 \* cos(α)

Вращение точки (x0, y0, z0) вокруг OY на угол α:

x = x0 \* cos(α) + z0 \* sin(α) (3)

y = y0

z = - x0 \* sin(α) + z0 \* cos(α)

## 2.3 Сдвиг сцены

Сдвиг сцены осуществляется элементарным суммированием координат каждой вершины со значениями сдвига из камеры dx, dy и dz:

x = x0 + dx (4)

y = y0 + dy

z = y0 + dz

## 2.4 Перспективные преобразования

Формируем матрицу перспективного проецирования:

****

где – коэффициент поля зрения

После умножения точки (x, y, z, 1) на матрицу перспективного проецирования, получим (x, y, z, z \* p).

Делим все компоненты точки на z \* p, чтобы масштабный коэффициент стал равным единице, и получим:

(, , , 1) (5)

## 2.5 Фильтрация поверхностей

### 2.5.1 Понятие фильтрации поверхностей

В данной программе для построения изображения мы используем алгоритм, использующий буфер глубины. Скорость его работы линейно зависит от числа граней, которые мы растеризуем, причём растеризация и определение глубины каждой точки растеризованной грани занимает большую часть времени работы алгоритма. Поэтому, если бы мы нашли способ сократить количество растеризуемых граней, то мы бы получили выигрыш во времени. Для решения этой задачи и используются алгоритмы фильтрации граней, позволяющие определить, будет ли видна текущая поверхность или нет, ещё до её растеризации и получения глубин всех её точек. В своей программе я использовал два алгоритма фильтрации: определение, является ли текущая грань лицевой, и решение вопроса о полной загороженности текущей грани другими поверхностями.

### 2.5.2 Фильтрация нелицевых граней

Представим, что у нас есть модель, которую необходимо отрисовать на экране с помощью алгоритма, использующего буфер глубины, - тетраэдр SABC. И расположен он по отношению к наблюдателю, как показано на рисунке 4.

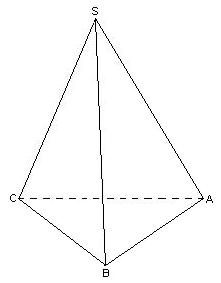


Рисунок 4

Очевидно, что будут отрисованы лишь грани SCB и SAB, потому что они лицевые. Грани же ABC и SCA отображены не будут, так как они не являются лицевыми. Однако в классической реализации алгоритма с z-буфером эти нелицевые грани тоже будут растеризованы, на что будет затрачено немало времени. Очевидным решением проблемы значительной потери времени здесь будет не растеризовывать нелицевые грани. Для этого необходимо определить, какие грани являются лицевыми, а какие нет.

Алгоритм решения этой задачи представлен на рисунке 5.

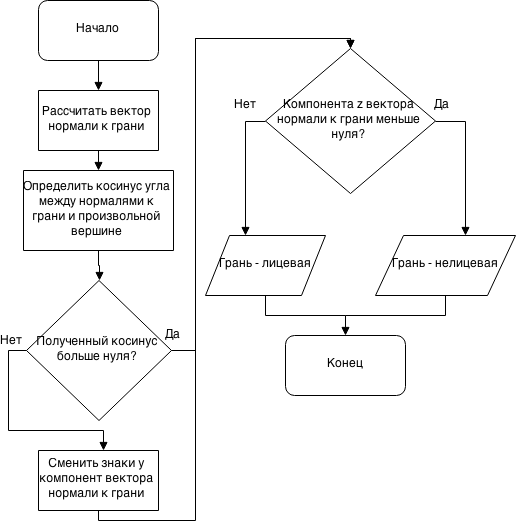


Рисунок 5

### 2.5.3 Фильтрация заведомо полностью загороженных граней

Другим способом фильтрации является недопущение растеризации граней, уже заведомо скрытых за другими. Пример подобной ситуации представлен на рисунке 6. Поверхность EFGH полностью загорожена ABCD.

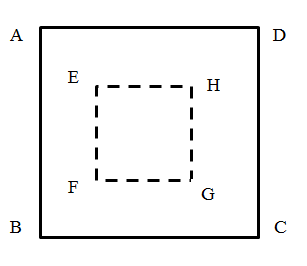


Рисунок 6

Для того, чтобы понять, нарисованы ли уже поверхности впереди данной, необходимо сравнить значения координат z каждой вершины текущей грани с соответствующими им значениями в z-буфере. Если все вершины поверхности меньше уже заложенных в буфер глубины значений, то это означает, что данная грань скорее всего уже загорожена другими. Однако это ещё не даёт абсолютной гарантии, так как возможны редкие случаи, когда разные поверхности загораживают лишь вершины рассматриваемой, делая её таким образом частично видимой. Однако чем меньше размер данной поверхности, тем меньше вероятность такой ситуации, а в случае её возникновения, с уменьшением размеров, это становится всё менее заметно для человеческого глаза, и таким образом можно найти компромисс. Решение данной проблемы рассмотрено в исследовательской части данной работы.

## 2.6 Совмещение алгоритма, использующего z-буфер, и закраски Гуро

Как уже упоминалось в пункте 2.3, алгоритм, использующий z-буфер, и закраска методом Гуро требуют большого количества интерполяционных вычислений. Причём ведутся они по одним и тем же сканирующим строчкам Y в одинаковых пределах X. Только в одном случае вычисляется глубина z, а в другом интенсивность света в текущем канале. А раз алгоритмы имеют общие места, их можно объединить для повышения производительности программы. В таком случае формальное описание нового алгоритма будет выглядеть следующим образом:

1. Заполнить буфер кадра фоновым значением цвета

2. Заполнить Z - буфер минимальным значением глубины Z

*3. Для каждого треугольника*

3.1.Преобразовать текущий треугольник в растровую форму.

*3.2. Для каждого пикселя текущего треугольника*

3.2.1.Вычислить глубину пикселя z(x,y).

3.2.2 . Сравнить z(x,y) со значением Z - буфер(x,y), хранящимся в Z – буфере в той же позиции.

*3.2.3. Если z(x,y) > Zбуфер(x,y)*

Zбуфер(x,y) = z(x,y).

*3.2.4 Для каждого из цветовых каналов*

3.2.4.1. Вычислить интенсивность I(x,y)

3.2.4.2. Занести значение I в соответствующий канал искомого пикселя в буфере кадра.

*конец цикла по пикселям треугольника (3.2)*

*конец цикла по треугольникам (3)*

## 2.7 Распараллеливание

Для повышения производительности программы можно выполнять действия, связанные с разными областями памяти, в разных потоках. Под этот критерий подходят геометрические преобразования моделей сцены.

В первую очередь программным путём выясняется количество ядер процессора на текущей машине, и в зависимости от этой цифры определяется количество потоков, между которыми делятся фигуры из сцены для их геометрических преобразований: поворота, сдвига и перспективного проецирования.

# 3 Технологический раздел

## 3.1 Выбор языка программирования

Выбор языка программирования и среды разработки для данного проекта обуславливается его спецификой. Во-первых, это некоммерческий проект с открытым кодом. Во-вторых, проект состоит из множества сложных математических вычислений и операций с памятью, что требует мощного аппарата адресной арифметики для повышения производительности. Также из-за большого количество пользовательских классов, для повышения надёжности продукта, желательна строгая типизация языка. Всем этим условиям удовлетворяет язык C++, который и был выбран для данного проекта. В качестве библиотеки разработки пользовательского интерфейса был выбран кроссплатформенный фреймворк с открытым кодом Qt.

## 3.2 Выбор формата файлов описания объектов сцены

В качестве спецификации файлов объектов для данного проекта был выбран популярный формат описания трёхмерных объектов OBJ (Wavefront). Описание моделей в такой спецификации состоит из 2ух файлов с расширением \*.obj и \*.mtl соответственно. В файле \*.obj содержатся геометрические данные о модели, а в файле \*.mtl данные о физических параметрах поверхностей модели. Этот формат полностью удовлетворяет спецификации, заданной в конструкторском разделе, пункте 3.1. Подробнее об этом формате можно прочитать в источниках [4] и [5].

## 3.3 Архитектура программы

### 3.3.1 Общая структура и формы

* Программа имеет три формы Главная форма с окном обозревателя
* Форма редактора интерьера
  + Окно плана комнаты
  + Каталог мебели
  + Список объектов на сцене
* Форма заполнения размеров сцены

Все три формы поддерживают любое разрешение и его динамическую смену.

### 3.3.2 Общий порядок работы с программой

Общий алгоритм работы приложения с точки зрения её внешнего функционала можно описать следующим образом:

1. Запрос на создание плана сцены / открытие существующего
2. Открытие окна редактора
3. Обход директорий на диске, выделенных под хранение файлов моделей
4. Формирование каталога доступных моделей
5. Добавление объектов на план сцены
6. Редактирование плана сцены (перенос объектов, удаление)
7. Передача плана сцены обозревателю
8. Загрузка моделей, описанных в плане сцены из файлов
9. Формирование сцены
10. Передача сцены в модуль рисования
11. Создание изображения
12. Передача изображения в обозреватель
13. Отображение изображения на экране
14. Взаимодействие с пользователем: запросы на поворот, приближение и отдаление камеры.
15. Сохранение сцены

### 3.3.3 Структуры данных

1) Класс сцены на рисунке 7

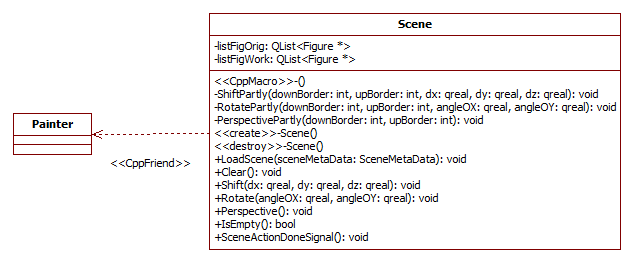


Рисунок 7

2) Классы моделей на рисунке 8

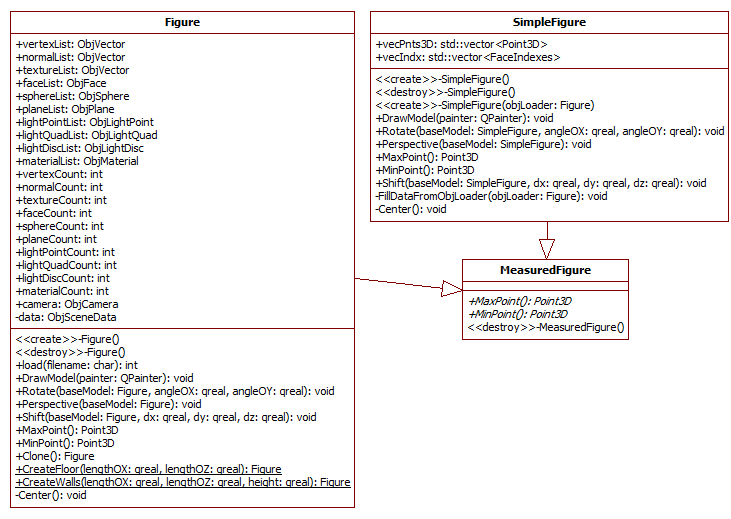


Рисунок 8

Класс Figure используется для полной поддержки формата Obj и рисования, а SimpleFigure для фоновых вычислений максимальных и минимальных точек.

3) Класс камеры на рисунке 9

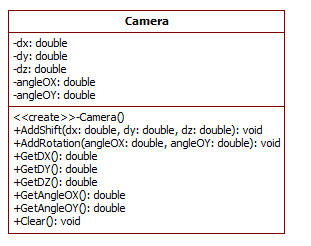


Рисунок 9

4) Классы поверхности и материала на рисунке 10

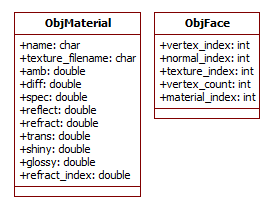


Рисунок 10

### 3.3.4 Применение паттернов проектирования

Для обеспечения стабильной работы программы, а также гибкости в её разработке и удобстве в дальнейшем сопровождении используются широко распространённые паттерны (шаблоны) проектирования [6].

Данная программа спроектирована с применением следующих паттернов:

* Композитная команда (Composite Command) [6]
* Фасад (Facade) [6]
* Одиночка (Singleton) [6]
* Стратегия (Strategy) [6]

### 3.3.5 Паттерн Команда (Command)

Паттерн команда используется для разделения интерфейса и домена вычислений. Как правило, команды связывают обработчики событий пользовательского интерфейса и соответствующую реакцию на них в фасаде (если он имеется). Иногда применяется композитная команда, позволяющая создавать очередь событий, что и было реализовано в данной программе. Диаграмму классов, связанную с командой можно увидеть на рисунке 11.

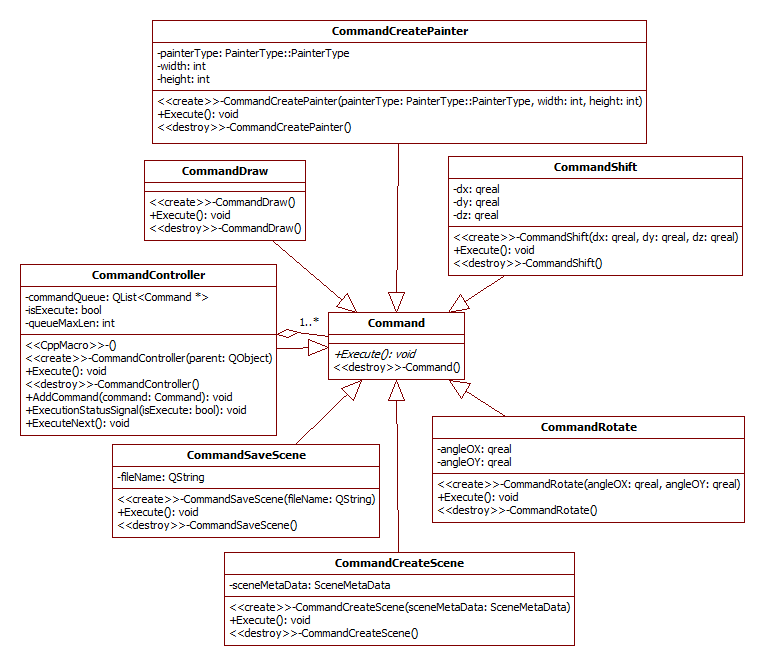


Рисунок 11

### 3.3.6 Паттерн Одиночка (Singleton)

Паттерн Одиночка позволяет иметь всего один экземпляр определённого класса во всей программе. В данном случае уместно применить его к фасаду. Диаграмма классов изображена на рисунке 12.

### 3.3.6 Паттерн Фасад (Facade)

Паттерн Фасад применяется для агрегирования и композиции всех важных составляющих частей программы, и тем самым обеспечивая централизованный доступ к ним и их слаженную работу и взаимодействие.

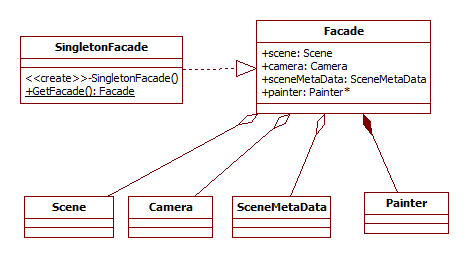


Рисунок 12

### 3.3.7 Паттерн Стратегия

Паттерн стратегия используется в случаях, когда необходимо во время выполнения программы подменять один алгоритм другим, за счёт инкапсуляции алгоритма в полиморфный класс. В данной программе применяется для подмены алгоритмов отрисовки, смотри рисунок 13

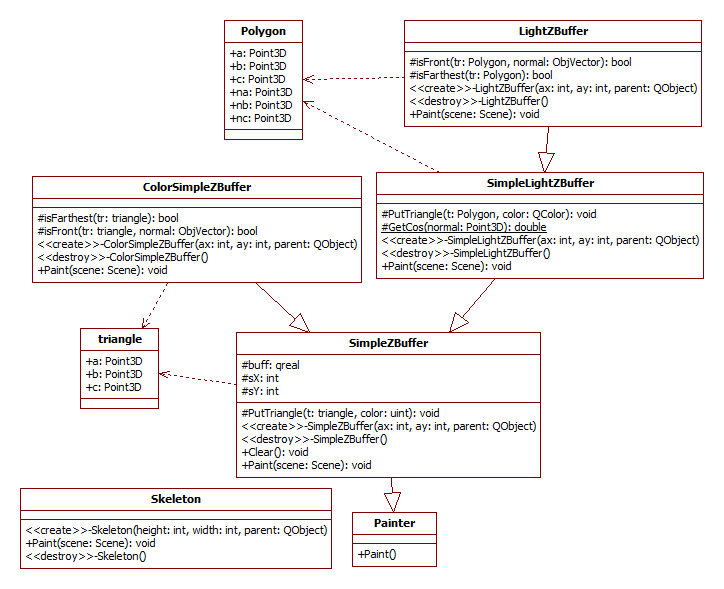


Рисунок 13

## 3.4 Интерфейс программы

### 3.4.1 Форма введения размеров сцены

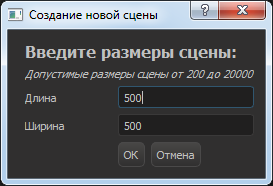


Рисунок 14

### 3.4.2 Форма редактора интерьера

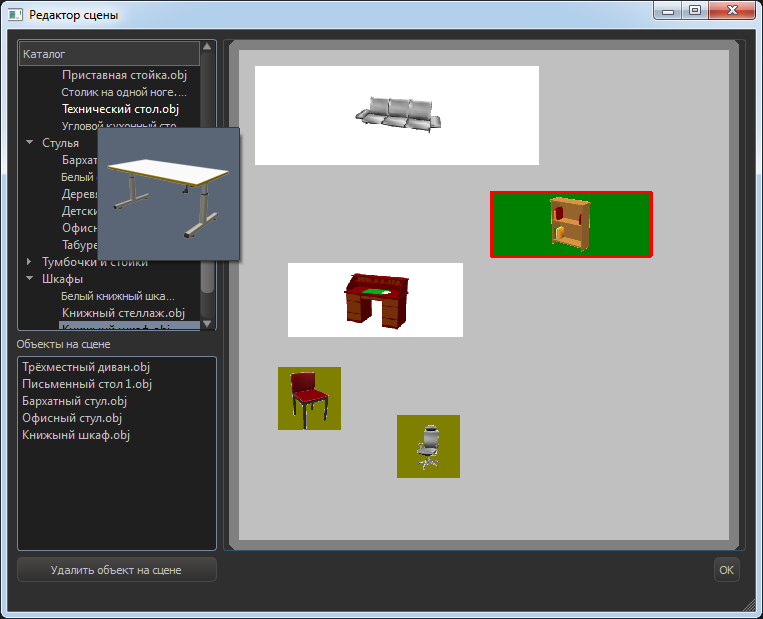


Рисунок 15

### 3.4.3 Форма обозревателя комнаты

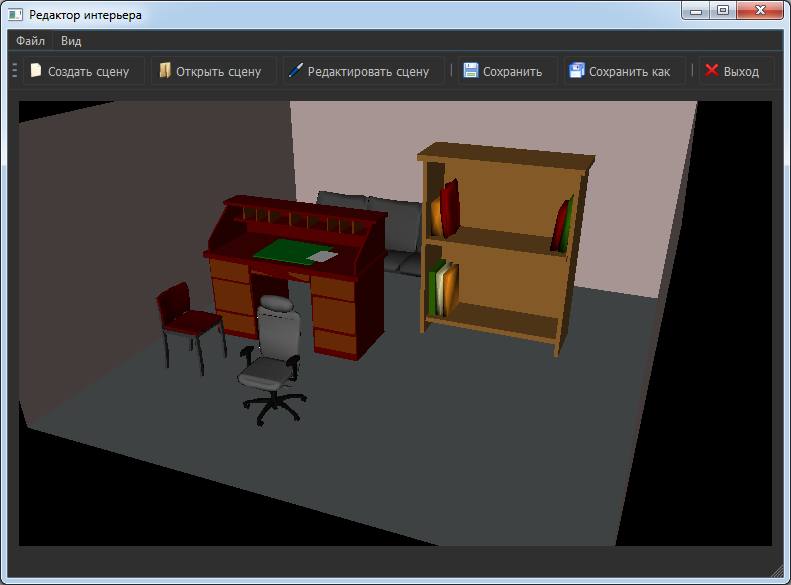


Рисунок 16

# 4 Исследовательская часть

## 4.1 Фильтрация уже полностью загороженных граней

В пункте 3.5.3 описывался способ фильтрации уже загороженных граней. Так же упоминалась проблема его применения из-за возможных погрешностей. Но, как выяснилось при разработке программы, возможно найти приемлемый компромисс, обеспечивающий выполнение данного метода при соблюдении некоторых условий, и сводящий ошибки к минимуму.

Условиями, при выполнении которых, можно использовать данный способ, являются длины проекций грани на оси X и Y не больше некоторой константы. Эмпирическим путем была установлена данная константа для моей программы, равная 30.

## 4.2 Временн*а*я оценка увеличения производительности за счёт фильтраций

Также были произведены тесты, результаты которых позволили определить выигрыш от применения фильтраций в конкретных числах.

В таблице по оси X отложено количество полигонов, а по Y – среднее количество кадров в секунду.

Тестирование проводилось на компьютере со следующими характеристиками:

* CPU Intel i7
* RAM 8192 MB

Как видно из таблицы ниже, фильтрации оправдали своё применение.

Кадры в секунду

Кол-во граней

# Заключение

Разработан программный продукт в соответствии с поставленным техническим заданием: реализованы возможность создания комнаты любого наполнения и любой сложности сцены, сохранение и загрузка сцены лабиринта из файла, возможность редактирования сцены. Кроме того, реализованы гибкие механизмы подмены алгоритмов прямо во время выполнения программы.

В процессе разработки проведен анализ предметной области: изучены алгоритмы удаления невидимых линий и поверхностей, модели освещения, алгоритмы наложения текстур.

Была проведена оптимизация программы методом фильтраций, в результате которой была достигнута высокая производительность программы и получено приемлемое значение FPS, подтверждённая тестами.

# Список литературы

1. Куров А.В. Курс лекций по машинной графике. – М., 2013
2. Роджерс Д., Адамс Дж. Математические основы машинной графики. – М.: Мир, 2001.
3. Авдеева С.М., Куров А.В. Методические указания по выполнению курсовой работы по дисциплине «Компьютерная графика». – М., 1995.
4. Сайт http://www.martinreddy.net/gfx/3d/OBJ.spec
5. Сайт <http://www.fileformat.info/format/material/>
6. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Примеры объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2001.