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:**

Визуализация расстановки контейнеров в грузовом отсеке самолета.

Студент  **\_\_\_\_\_\_\_\_\_\_\_\_**  Зюрин М.В.

(Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта \_\_\_\_\_\_\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_ \_**Филлипов М.В.

(Подпись, дата) (И.О.Фамилия)

Оглавление

[Введение 4](#_Toc499754483)

[1 Аналитический раздел 5](#_Toc499754484)

[1.1 Распространённые алгоритмы удаления невидимых линий и поверхностей 5](#_Toc499754485)

[1.1.1 Алгоритм Робертса 5](#_Toc499754486)

[1.1.2 Алгоритм трассировки лучей 6](#_Toc499754487)

[1.1.3 Алгоритм Варнока 6](#_Toc499754488)

[1.1.4 Алгоритм, использующий z-буфер 7](#_Toc499754489)

[1.2 Распространённые алгоритмы закрасок 8](#_Toc499754490)

[1.2.1 Модель освещения Ламберта 8](#_Toc499754491)

[1.2.2 Затенение по Гуро 9](#_Toc499754492)

[1.2.3 Закраска по Фонгу 10](#_Toc499754493)

[1.2.4. UV-преобразование 11](#_Toc499754494)

[1.3 Выводы и выбор алгоритмов для поставленной задачи 12](#_Toc499754495)

[2 Конструкторский раздел 12](#_Toc499754496)

[2.1 Структуры данных 12](#_Toc499754497)

[2.2 Общий алгоритм построения изображения в программе 13](#_Toc499754498)

[2.2 Поворот сцены 15](#_Toc499754499)

[2.3 Сдвиг сцены 15](#_Toc499754500)

[2.4 Перспективные преобразования 16](#_Toc499754501)

[2.5 Фильтрация поверхностей 16](#_Toc499754502)

[2.5.1 Понятие фильтрации поверхностей 16](#_Toc499754503)

[2.5.2 Фильтрация нелицевых граней 17](#_Toc499754504)

[2.5.3 Фильтрация заведомо полностью загороженных граней 18](#_Toc499754505)

[3 Технологический раздел 20](#_Toc499754506)

[3.1 Выбор языка программирования 20](#_Toc499754507)

[3.2 Выбор формата файлов для описания объектов сцены 20](#_Toc499754508)

[3.3 Архитектура программы 21](#_Toc499754509)

[3.3.1 Общая структура и формы 21](#_Toc499754510)

[3.3.2 Общий порядок работы с программой 21](#_Toc499754511)

[3.3.3 Структуры данных 21](#_Toc499754512)

[3.3.4 Применение паттернов проектирования 23](#_Toc499754513)

[3.3.5 Паттерн Декоратор (Decorator) 23](#_Toc499754514)

[3.3.6 Паттерн Подписчик-издатель (Publisher-subscriber) 24](#_Toc499754515)

[3.3.6 Паттерн Фасад (Facade) 24](#_Toc499754516)

[3.3.7 Паттерн Обещание (Promise) 25](#_Toc499754517)

[3.4 Интерфейс программы 26](#_Toc499754518)

[3.4.1 Панель управления 26](#_Toc499754519)

[3.4.2 Веб-страница 26](#_Toc499754520)

[Заключение 27](#_Toc499754521)

[Список литературы 28](#_Toc499754522)

# 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 расстановки контейнеров в грузовом отсеке самолета.

Программа представляет из себя веб-приложение, получающее файл с заранее рассчитанными координатами контейнеров и отображающее расстановку в браузере. Предусмотрена интерактивность в виде поворота камеры, приближения и отдаления объектов сцены, возможность скрыть/отобразить корпус самолета.

# 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.

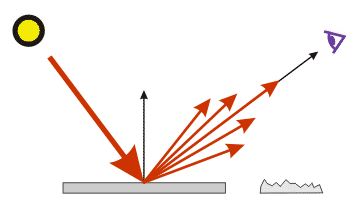


Рисунок 1

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.

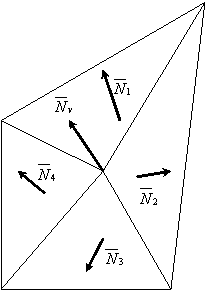


Рисунок 2

### 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.

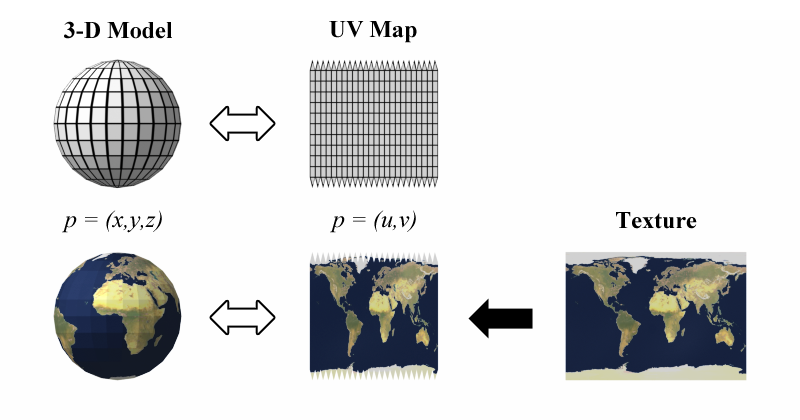
Основным главными недостатками данного алгоритма являются ресурсоемкость и большая вычислительная сложность.

### 1.2.4. UV-преобразование

UV-преобразование или развёртка в трёхмерной графике (UV map) — соответствие между координатами на поверхности трёхмерного объекта (X, Y, Z) и координатами на текстуре (U, V). Значения U и V обычно изменяются от 0 до 1. Развёртка может строиться как вручную, так и автоматически.

Современное трёхмерное аппаратное обеспечение считает, что UV-преобразование в пределах одного треугольника является аффинным — поэтому достаточно задать U и V для каждой вершины каждого из треугольников. Существует несколько противоречащих друг другу показателей качества развёртки:

* Максимально полное использование площади текстуры. Впрочем, в зависимости от разрыва между «минимальными» и «максимальными» системными требованиями, по краям развёртки текстуре нужен определённый «припуск» на генерацию текстур меньшего размера.
* Отсутствие областей с недостаточной или избыточной детализацией текстуры.
* Отсутствие областей с излишними геометрическими искажениями.
* Сходство со стандартными ракурсами, с которых обычно рисуется или фотографируется объект.



**Рисунок 3**

## 1.3 Выводы и выбор алгоритмов для поставленной задачи

В программе предвидится большое количество объектов на сцене. Также для создания удобной работы требуется высокая скорость отрисовки, близкая к реальному времени, поэтому необходим быстродействующий алгоритм, способный работать с большим количеством граней и не требовать больших затрат ресурсов компьютера, т.к. подразумевается, что использовать данный продукт будут пользователи веб-сайта авиакомпании. Под эту роль хорошо подходит алгоритм, использующий буфер глубины. Для создания изображения, близкого к реалистичному необходимо использование одной из моделей освещения и закраски. Так как главное в этом проекте – быстродействие и производительность, то самым рациональным решением, на мой взгляд, является использование модели освещения Ламберта и UV-преобразования для наложения текстуры на геометрии.

# 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 и экземпляр камеры

2) Модель включает следующие данные:

* Массив вершин фигуры
* Массив векторов нормалей к вершинам
* Вершины UV-проекции для наложения текстуры
* Текстура

Данные модели:

1. Вершина и нормаль представляют собой три вещественных числа
2. Текстура представляет собой изображение в формате .png, которое в процессе работы алгоритма будет «натягиваться» на геометрию модели посредством UV-проецирования.
3. Камера содержит:

* Положение в пространстве, заданное 3 вещественными числами X, Y, Z
* Матрица проекции и видовая матрица

## 2.2 Общий алгоритм построения изображения в программе

Вход: Файл расстановки грузовых контейнеров

Выход: изображение (заполненный буфер кадра)

Алгоритм представлен на рисунке 3.

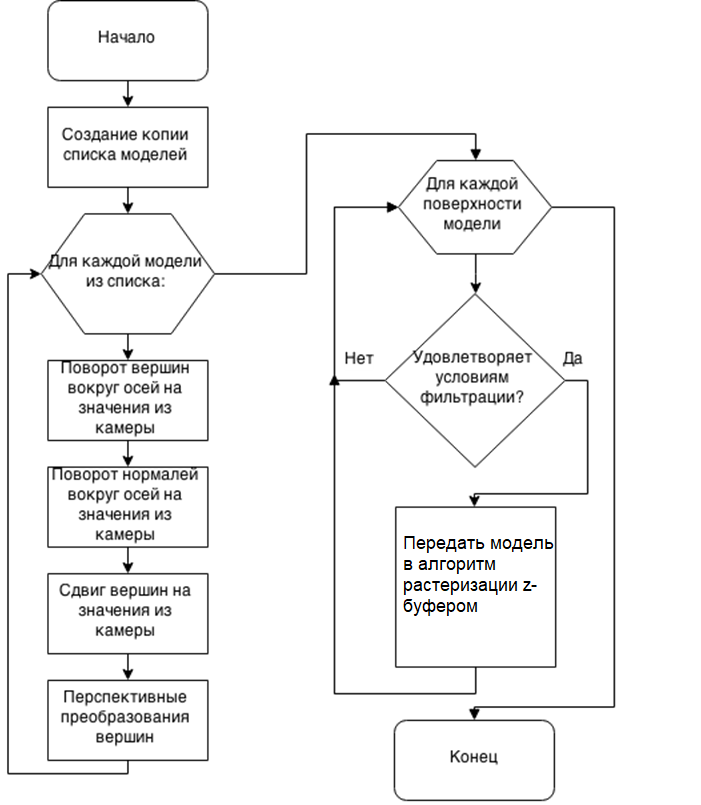


Рисунок 4

## 2.2 Поворот сцены

В данном проекте реализована полное интерактивное взаимодействие пользователя со сценой. Предусмотрено вращение и приближение/отдаление камеры относительно объектов сцены.

Для задачи просмотра расстановки используются повороты по всем осям координат. 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.

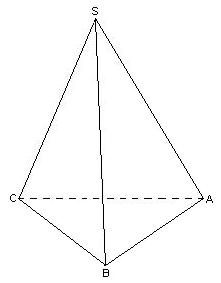


Рисунок 5

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 6.

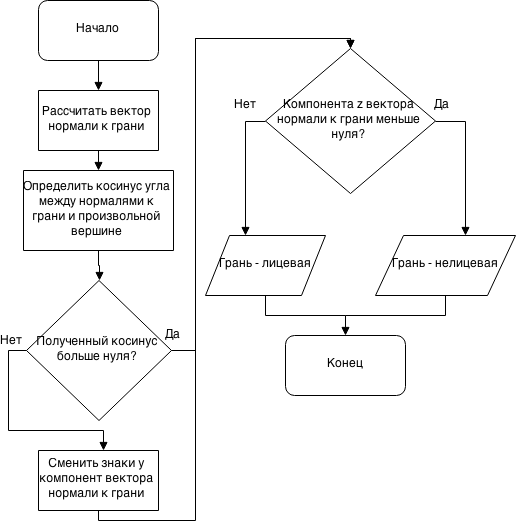


Рисунок 6

### 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 7. Поверхность EFGH полностью загорожена ABCD.

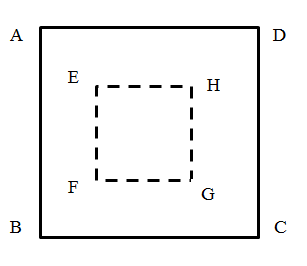


Рисунок 7

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.

# 3 Технологический раздел

## 3.1 Выбор языка программирования

Выбор языка программирования и среды разработки для данного проекта обуславливается его спецификой. Во-первых, данный проект является частью большой веб-системы обработки заказов клиентов по осуществлению авиа-перевозок. Во-вторых, проект должен быть не требователен к системе, на которой он выполняется. Проект изначально задумывается кроссплатформенным. Он должен выполняться одинаково, как на мобильном устройстве, так и на ПК. Чтобы проект мог отвечать всем этим условиям, выбор пал на разработку frontend компонента данной веб-системы на фреймворке Angular. В качестве языка программирования был выбран язык TypeScript – язык, разработанный корпорацией Microsoft на базе языка JavaScript. В языке реализован синтаксис для комфортной разработки веб-приложений в стиле ООП.

## 3.2 Выбор формата файлов для 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 файла с расширением \*.obj соответственно. В файле \*.obj содержатся геометрические данные о модели. Текстура, накладываемая поверх геометрии, хранится в виде изображения в формате .png. 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 представляет из себя веб-страницу, на которой размещено окно

отображения и панель управления, расположенная слева.

Форма поддерживает любое разрешение и его динамическую смену.

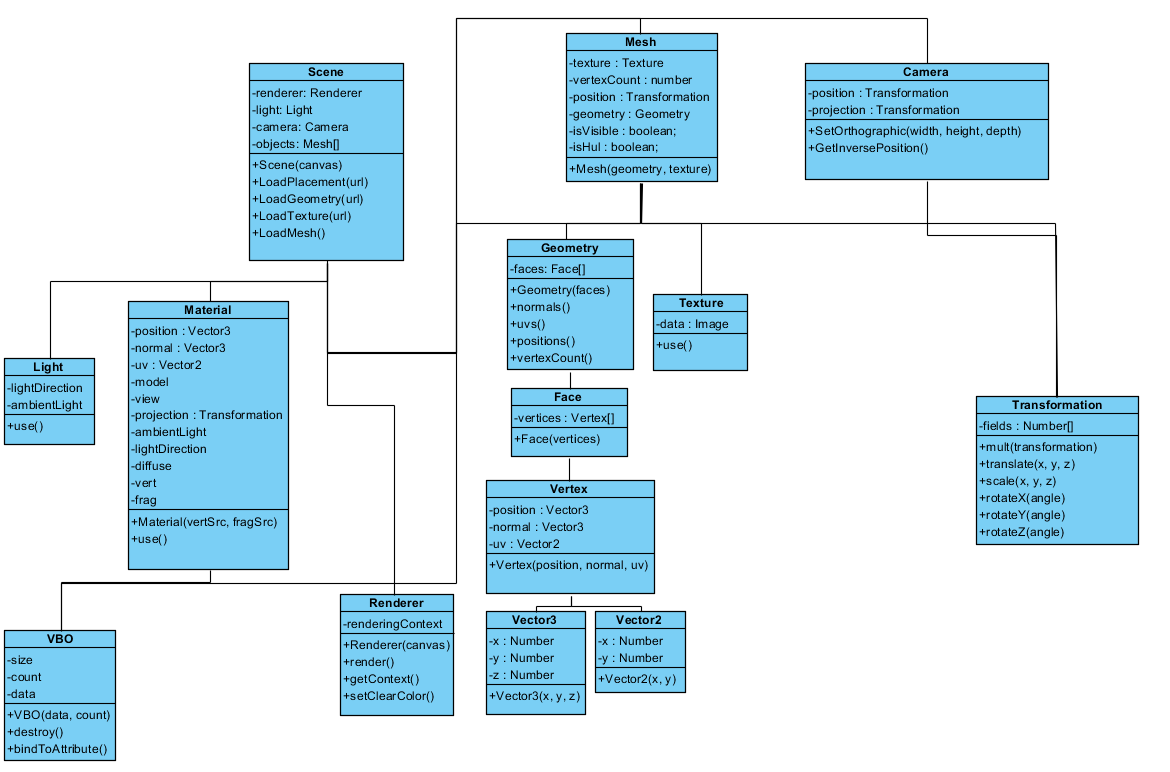
### 3.3.2 Общий порядок работы с программой

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

1. Загрузка веб-страницы
2. Выбор из выпадающего списка нужного файла расстановки контейнеров
3. Подтверждение выбора и отправка файла на отрисовку нажатием на соответствующую кнопку
4. Добавление объектов на сцену
5. Рендер изображения на окно отображения
6. Взаимодействие с пользователем: запросы на поворот, приближение и отдаление камеры, скрытие/отображения корпуса самолета.

### 3.3.3 Структуры данных

Классовая диаграмма проекта на рисунке 8:



Здесь Mesh является классом, содержащим данные о каждой модели сцены.

Класс Geometry описывает все грани одной модели, нормали этих граней и вершины UV-преобразования для наложения текстуры.

Новый Mesh создается после того, как были созданы соответствующие ему экземпляры Geometry и Texture.

Класс Scene содержит в себе один экземпляр класс Camera и массив объектов сцены. Методы класса Scene обеспечивают парсинг приходящих с сервера файлов и создание объектов.

### 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:

* Декоратор (Decorator) [6]
* Фасад (Facade) [6]
* Подписчик-издатель (Publisher-subscriber) [6]
* Обещание (Promise) [6]

### 3.3.5 Паттерн Декоратор (Decorator)

Паттерн декоратор используется для обращения к полям HTML элементов на верстке веб-страницы в основном коде программы (класс ElementRef фреймворка Angular).

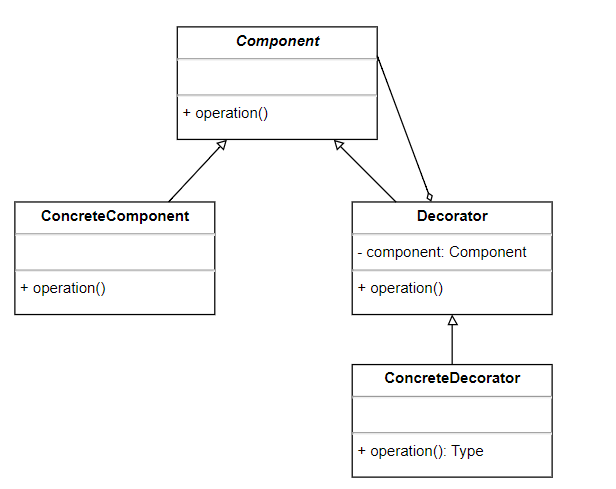


Рисунок 12

### 3.3.6 Паттерн Подписчик-издатель (Publisher-subscriber)

Паттерн Подписчик-издатель используется в программе для обработки запросов пользователя на поворот камеры и ее приближение/отдаление от объектов сцены. Также данный паттерн используется для загрузки необходимых файлов с сервера (расстановки, .obj и .png).

### 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.

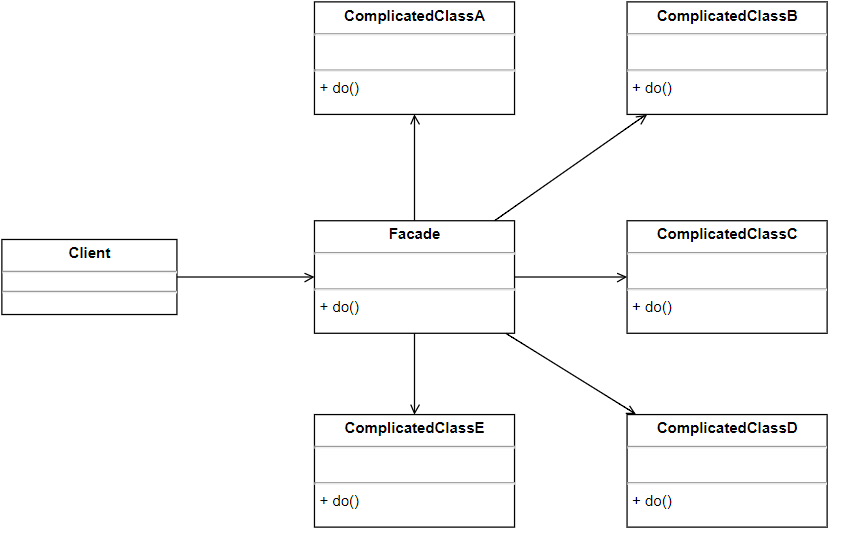


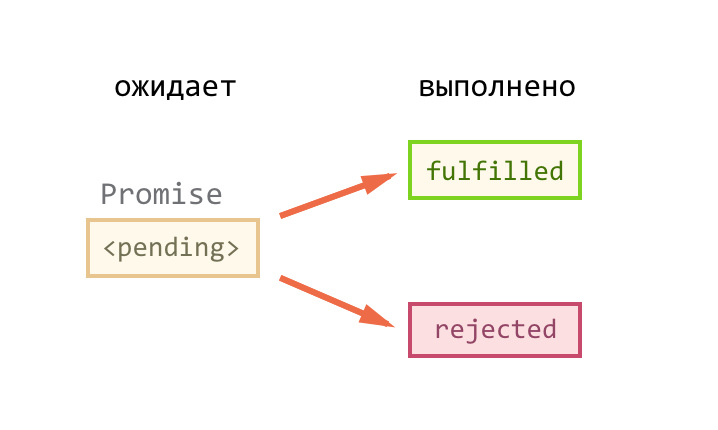
Рисунок 14

В данном проекте роль фасада выполняет класс сцены, который агрерирует в себе все составляющие графики и предоставляет интерфейс непосредственно пользователю (через компонент Angular).

### 3.3.7 Паттерн Обещание (Promise)

Паттерн Обещание используется для правильной организации загрузки необходимых файлов с сервера. Так как в будущем проект будет иметь backend составляющую в виде сервера, то аппарат загрузки и парсинга файлов посредством GET запросов необходимо на раннем этапе разработки.

В данном проекте используется JavaScript Promise для обеспечения загрузки файлов расстановки и моделей с корневой папки на сервере. Promise – это специальный объект, который содержит своё состояние. Вначале pending («ожидание»), затем – одно из: fulfilled («выполнено успешно») или rejected («выполнено с ошибкой»).



Cуществует два типа обработчиков promise:

1. onFulfilled – срабатывают, когда promise в состоянии «выполнен успешно».
2. onRejected – срабатывают, когда promise в состоянии «выполнен с ошибкой».

Способ использования:

Код, которому надо сделать что-то асинхронно, создаёт объект promise и возвращает его.

По завершении процесса асинхронный код переводит promise в состояние fulfilled (с результатом) или rejected (с ошибкой). При этом автоматически вызываются соответствующие обработчики во внешнем коде.

## 3.4 Интерфейс программы

### 3.4.1 Панель управления

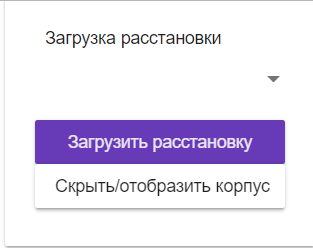
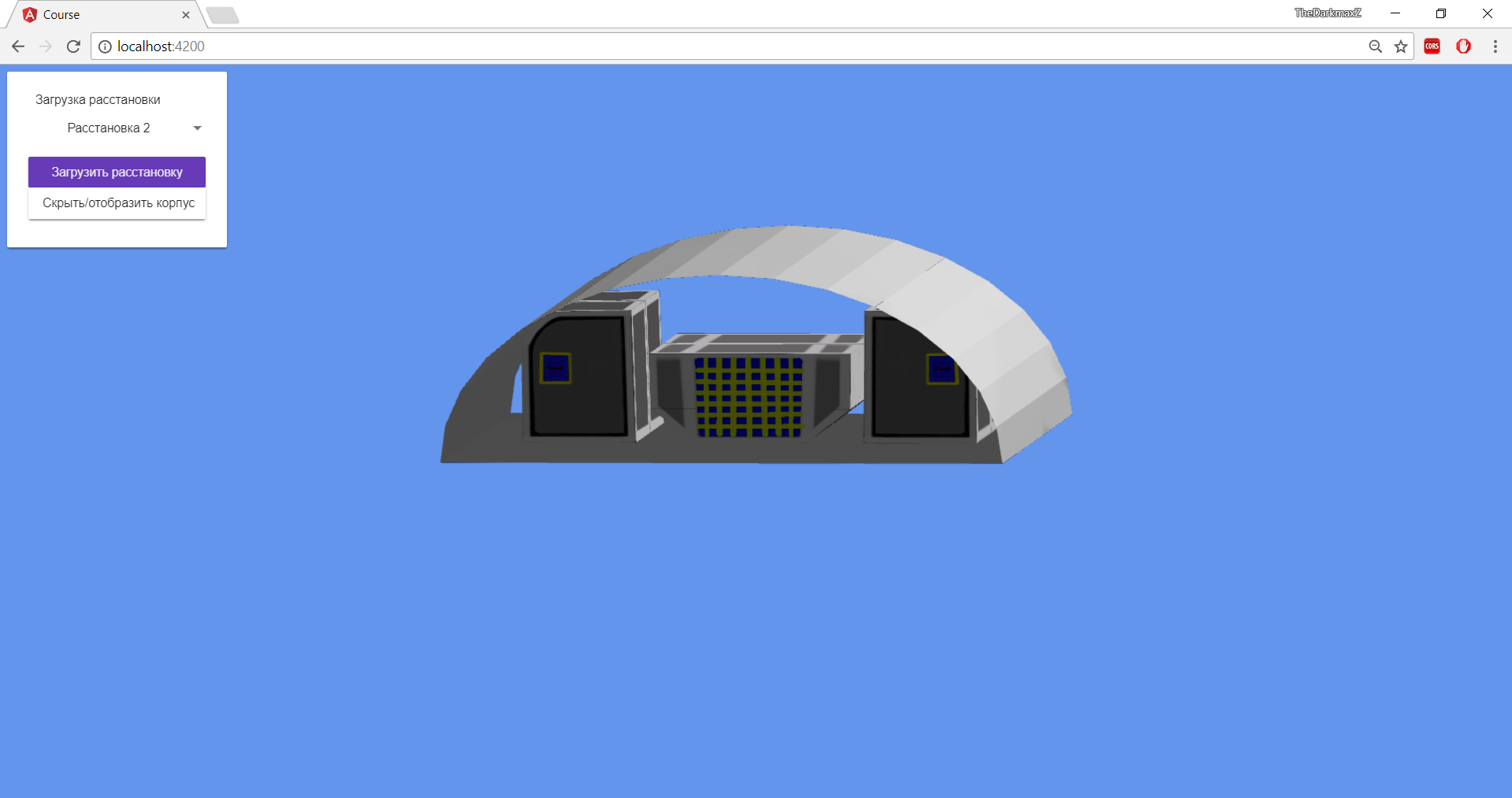


Рисунок 16

### 3.4.2 Веб-страница

Рисунок 17

# Заключение

Разработан программный продукт в соответствии с поставленным техническим заданием: реализованы веб-компонент отображения расстановки контейнеров в грузовом отсеке самолета. Все необходимые файлы передаются на клиент с сервера.

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 быстродействие и производительность.

# Список литературы

1. Куров А.В. Курс лекций по машинной графике. – М., 2017
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.
7. Официальный сайт фреймворка Angular <https://angular.io/>
8. Библиотека оформления веб-интерфейса Angular Material <https://material.angular.io/>