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:**

Реализация алгоритма тесселяции трёхмерного реалистичного объекта

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы \_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Москва, 20\_\_\_

Содержание

[Введение 3](#_Toc468923114)

[1. Аналитическая часть 5](#_Toc468923115)

[1.1 Выбор модели представления данных 5](#_Toc468923116)

[1.2 Выбор алгоритма отображения объектов 5](#_Toc468923117)

[1.3 Результаты аналитической части 7](#_Toc468923118)

[2. Конструкторская часть 9](#_Toc468923119)

[2.1 Описание концепции алгоритма тесселяции 9](#_Toc468923120)

[2.2 Описание входных данных 12](#_Toc468923121)

[2.3 Описание трёхмерных афинных преобразований 12](#_Toc468923122)

[2.4 Описание алгоритмов произвольных преобразований камеры 13](#_Toc468923123)

[3.Технологическая часть 17](#_Toc468923124)

[3.1 Выбор и обоснование языка и среды программирования 17](#_Toc468923125)

[3.2 Описание структуры программы 17](#_Toc468923126)

[3.3 Описание интерфейса программы 19](#_Toc468923127)

[4 Экспериментально-исследовательская часть 21](#_Toc468923128)

[4.1 Характеристики оборудования 21](#_Toc468923129)

[4.2 Результаты исследования 21](#_Toc468923130)

[Заключение 24](#_Toc468923131)

[Список использованной литературы 25](#_Toc468923132)

# Введение

Вопрос создания реалистичного изображения в компьютерных играх и 3D фильмах неоднократно поднимался на протяжении многих лет. Казалось бы, неуклонный рост вычислительной мощности графических процессоров давно должен был обеспечить фотореалистичные 3D-изображения на экранах наших мониторов. Но, несмотря на все обещания, до сих пор очень мало приложений способных обеспечить такую графику. Чем из большего количества полигонов состоит геометрическая модель, тем более детализированной и, как следствие, более реалистичной она кажется, но при этом для отображения такой модели требуется мощная аппаратная составляющая компьютера. Как одним из вариантов для упрощения вычислений и рендеринга изображения, близкого к фотореалистичному, служит алгоритм тесселяции.

Но, к сожалению в данное время не существует такого алгоритма, способного изменять полигональность модели динамически. В компьютерных играх данную проблему обходят наличием нескольких моделей объекта с различным числом полигонов, которые подгружаются в зависимости от удаления камеры от данного объекта. В 3D фильмах, наоборот имеют возможность использовать алгоритмы тесселяции, но в таком случае совершенно не важно, как быстро произойдет перерасчет числа полигонов объекта, так как в конечном итоге важен только текущий кадр.

Что же такое тесселяция? На базовом уровне тесселяция – это метод увеличения или уменьшения числа полигонов в объекте. К примеру, если вы возьмете квадрат и разрежете по диагонали, вы «тесселируете» этот квадрат в два треугольника. Сама по себе тесселяция не сильно улучшает реализм. К примеру, ничего не меняется, если в компьютерной игре квадрат визуализируется как два треугольника или как две тысячи треугольников. Тесселяция повышает реализм в том случае, если новые треугольники используются для описания новой информации.

За основу тесселяции взята идея о том, что при увеличении расстояния от объекта, с пропорциональным ей уменьшением детализации, внешний вид объекта останется прежним. Иными словами, если удалить камеру от объекта втрое дальше начального расстояния, при этом уменьшив число полигонов на объекте в три раза, внешний вид объекта визуально останется прежним.

В данном проекте допущено отступление от этого правила, что сделано из соображений наглядности. В этой программе число полигонов меняется без отдаления или приближения к объекту.

Для достижения поставленной цели нужно было решить следующие задачи:

* Изучить способы хранения дополнительной информации об объекте с целью выявить оптимальный (в контексте поставленной задачи).
* Разработать наиболее скоростной алгоритм, решающий задачу тесселяции для случайного трехмерного реалистичного объекта. Оценка производится по количеству кадров в секунду
* Выбрать модели данных, наилучшим образом облегчающие и ускоряющие процесс тесселяции.

# Аналитическая часть

* 1. Выбор модели представления данных

Был произведен анализ различных алгоритмов отображения сцены и аппроксимации объектов на ней. Вот 2 наиболее распространенных ныне способа представления объектов:

1. Представление объектов множеством полигонов (в общем случае – множеством треугольников), приблизительно описывающих форму модели
2. Представление объектов в виде аналитической модели, т.е. расчет каждого пикселя и отображение его на экране

Второй способ имеет много достоинств, в том числе:

1. Возможность простого включения в работу алгоритмов освещения, текстурирования, добавление тени и т.п.
2. Получение наиболее реалистичного изображение вследствие отдельных расчетов для каждого пикселя.

Но, очевидно, этот способ не является оптимальным, т.к. целью работы является отображение полигонов и изменение их количества, что невозможно при описании аналитической модели. Кроме того, на практике представление модели множеством полигонов дает ощутимые преимущества:

1. Возможность изменять степень аппроксимации тела
2. Совместимость с большинством алгоритмов рендеринга
3. Масштабирование тела происходит без увеличения сложности расчетов
4. Возможность описания алгоритмов взаимодействия тел друг с другом
   1. Выбор алгоритма отображения объектов

После этого также были рассмотрены несколько способов отображения сцены и объектов на ней, среди которых 3 наиболее значимых:

1. Алгоритм удаления невидимых линий Робертса

Алгоритм Робертса представляет собой первое известное решение задачи об удалении невидимых линий. Это математически элегантный метод, работающий в объектном пространстве. Алгоритм прежде всего удаляет из каждого тела те ребра или грани, которые экранируются 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, экранируются этими телами. Поэтому вычислительная трудоемкость алгоритма Робертса растет теоретически, как квадрат числа объектов. Математические методы, используемые в этом алгоритме, просты, мощны и точны, однако, он работает только с выпуклыми телами, причем делает это достаточно долго из-за квадратичной сложности. Метод непригоден непосредственно для передачи падающих теней и других сложных визуальных эффектов.

1. Алгоритм, использующий Z-буфер

Алгоритм, использующий z-буфер, - это один из простейших алгоритмов удаления невидимых поверхностей. Впервые он был предложен Кэтмулом. Работает этот алгоритм в пространстве изображения. Идея z-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пикселя в пространстве изображения, z-буфер - это отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пикселя в пространстве изображения. В процессе работы глубина или значение z каждого нового пикселя, который нужно занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксель расположен впереди пикселя, находящегося в буфере кадра, то новый пиксель заносится в этот буфер и, кроме того, производится корректировка z-буфера новым значением z. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по х и у наибольшего значения функции z (х, у).

Главное преимущество алгоритма – его простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей. Сцены могут быть любой сложности. Поскольку габариты пространства изображения фиксированы, оценка вычислительной трудоемкости алгоритма не более чем линейна. Поскольку элементы сцены или картинки можно заносить в буфер кадра или в z-буфер в произвольном порядке, их не нужно предварительно сортировать по приоритету глубины. Поэтому экономится вычислительное время, затрачиваемое на сортировку по глубине.

Основной недостаток алгоритма - большой объем требуемой памяти. Если сцена подвергается видовому преобразованию и отсекается до фиксированного диапазона значений координат z, то можно использовать z-буфер с фиксированной точностью. Информацию о глубине нужно обрабатывать с большей точностью, чем координатную информацию на плоскости (х, y); обычно бывает достаточно 20-ти бит. Буфер кадра размером 512\*512\*24 бит в комбинации с z-буфером размером 512\*512\*20 бит требует почти 1.5 мегабайт памяти. Другой недостаток алгоритма z-буфера состоит в трудоемкости и высокой стоимости устранения лестничного эффекта, а также реализации эффектов прозрачности и просвечивания.

1. Алгоритм художника

Идея алгоритма художника состоит в том, что, используя упорядоченность граней по глубине, на экран выводятся с закраской грани, начиная с дальних. При этом грани, которые выводятся позже, закрывают собою невидимые части более дальних граней. То есть, пусть имеется некий набор граней (т.е. сцена), который требуется нарисовать. Отсортируем грани по удаленности от камеры и отрисуем все грани, начиная с самых удаленных. Довольно распространенная характеристика удаленности для грани ABC - это среднее значение z, mid\_z = (A.z+B.z+C.z)/3. Стоит отметить, что деление на 3 выполнять не обязательно, т.к. в алгоритме эта операция проводится для каждого многоугольника. Алгоритм очень прост, но он не лишен недостатков. Во-первых, при некотором расположении граней этот алгоритм вообще не может дать правильного результата - в каком порядке грани не рисуй, получится неправильно. Нужно разбивать каждый спорный полигон на несколько полигонов поменьше и работать с каждым в отдельности. И, во-вторых, при использовании этого алгоритма отрисовываются вообще все грани сцены, и при большом количестве загораживающих друг друга граней мы будем тратить большую часть времени на отрисовку невидимых в конечном итоге частей.

* 1. Результаты аналитической части

Из рассмотренных выше вариантов были выбраны наиболее оптимальные: представление модели в качестве множества полигонов (треугольников) и построение сцены с помощью алгоритма художника. Сделано это из тех соображений, что тесселяция подразумевает собой наличие полигонов и изменение их количества, поэтому представление в качестве аналитической модели здесь не подходит. Теоретически, это возможно, но в таком случае понятие полигона вводилось бы искусственно, что не сделало бы программу более понятной для пользователя и более простой для программиста. Также большую роль сыграла среда разработки. В ней при достаточно большом количестве расчетов, которые происходят при тесселяции случайного объекта, их время заметно превышает время отрисовки модели.

# Конструкторская часть

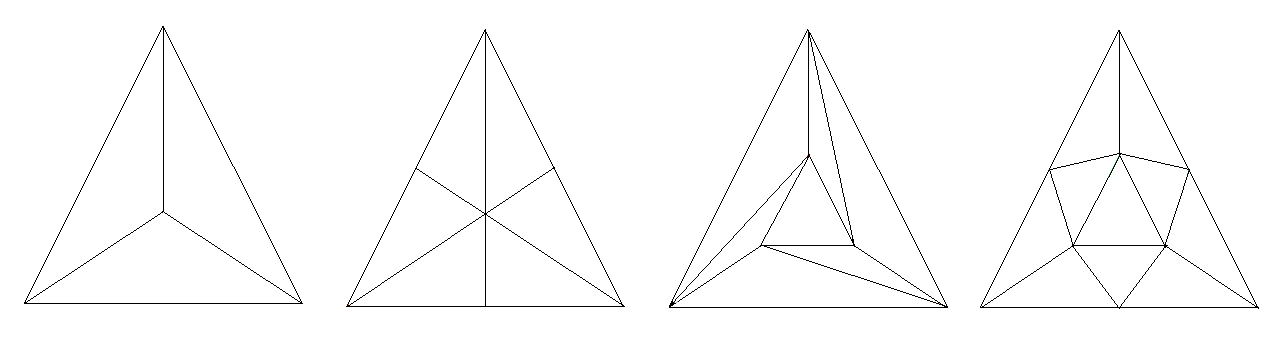
* 1. Описание концепции алгоритма тесселяции

Как уже упоминалось выше, тесселяция – изменение числа треугольников, составляющих объект, при его масштабировании. Соответственно, тесселяцию можно разбить на 2 части:

1. Увеличение числа полигонов.
2. Уменьшение числа полигонов.

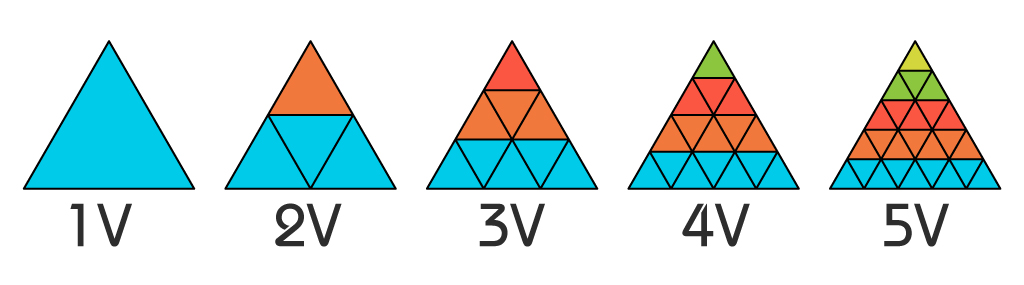
Сначала рассмотрим 1 случай. Понятие "частота триангуляции" подразумевает плотность разбивки объекта на треугольники. Т.е. один и тот же объект можно "описать" разным количеством треугольников. К примеру, для менее плотной разбивки потребуется меньше треугольников, но с большей длиной ребра и форма будет более угловатой. Для более плотной разбивки потребуется большее количество треугольников с меньшей длиной ребра, но форма получится боле ровной и близкой к сферической.

Существует огромное множество способов разбиения полигонов на более мелкие треугольники. Вот лишь некоторые из них:



Но все они имеют свои недостатки. Какие-то из способов требуют слишком много расчетов (№2 и №4), какие-то не сохраняют треугольники равносторонними (№1 и №3), поэтому было принято решение использовать вариант разбиения треугольника линиями, соединяющими центры его сторон. Этот способ устраняет оба недостатка, описанных выше.

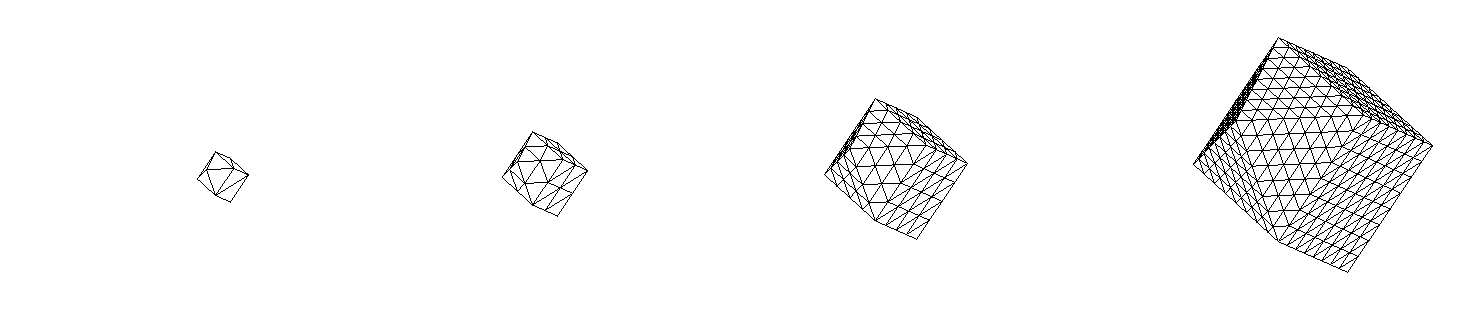
В мире используется стандартное обозначение частоты разбиения латинской буквой "V". Ниже приведены примеры триангуляции до пятого значения. Число значения частоты равняется количеству "рядов", на которые делится один из треугольников тесселируемого объекта.



Однако, если нам важна скорость работы, подобный подход в триангуляции полигонов может быть использован только при первой генерации модели. Переход от «2V» к «3V», от «3V» к «4V» и т.д. сложен с математической точки зрения. В этом случае нужно возвращаться к позиции «1V», триангуляция из которой наиболее проста (нужно взять середины сторон исходного треугольника и соединить их линиями). Очевидным образом мы приходим к оптимальному с точки зрения быстродействия алгоритму. При каждой тесселяции мы увеличиваем число треугольников в 4 раза. Т.е. после этапа «1V» идет этап «2V», за ним «4V», «8V» и так далее. Нам не нужно выполнять дополнительные расчеты, поэтому подобный алгоритм работает быстрее классического. Таким образом, каждому из N треугольников низкополигональной модели соответствуют

полигонов высокополигональной модели, где K – число тесселяций объекта при приближении из максимально далекой точки пространства в максимально близкую к наблюдателю.

Но увеличение числа треугольников бесполезно, если они не несут в себе новой информации. Форма модели и отсутствие острых углов очень важны для реализма трехмерного объекта. Вот пример тесселяции сферы треугольниками, которые не несут дополнительной информации об объекте:

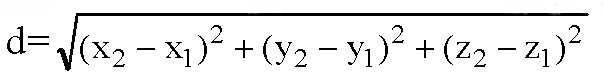


Этот объект гораздо больше похож на икосаэдр, чем на шар. Тесселированные треугольники ситуацию не улучшают. Способов исправить это 2:

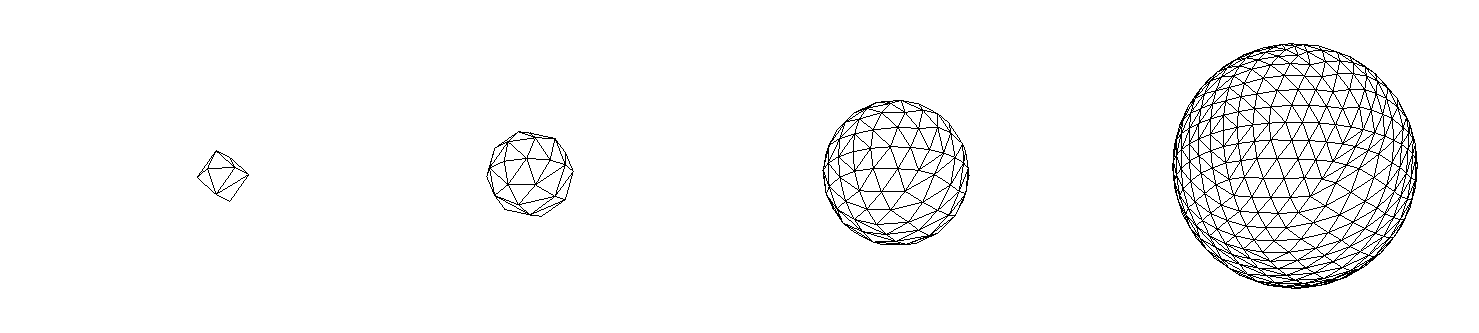
1. Использовать карты смещений (displacement mapping)
2. Использовать высокополигональную модель

Первый способ весьма сложен в реализации и не подходит для замкнутых объектов, гораздо лучше обрабатывая поверхности и ландшафты, вследствие чего в проекте используется высокополигональная модель. Она всегда загружена в программу, но не отображается. После триангуляции берутся координаты новых точек и ищутся ближайшие к ним в высокополигональной модели.

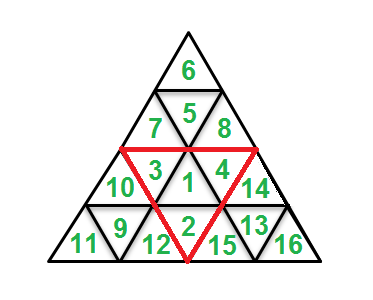
Поиск осуществляется по следующей формуле:



В результате мы видим значительный прогресс в реалистичности отображения



2 случай отдаления объектов и уменьшения число треугольников значительно проще. В программе данные хранятся таким образом, что в массиве полигонов первым идет центральный для какой-то из граней, а за ним – три примыкающих к нему полигона. Соответственно, можно просто удалить эти 4 полигона, записав на их место 1, образованный этими треугольниками.



Такой подход значительно уменьшает время выполнения тесселяции.

* 1. Описание входных данных

В данной программе входные данные подаются в виде файла с расширением .obj.

Это известный формат файлов в 3D моделировании. Он удобен своей внутренней структурой, которая подразумевает хранение треугольных полигонов, а также информации об их расположении в пространстве. Также в таком файле возможно передавать информацию о нормалях и текстурах, что пригодиться при расширении возможностей данной программы. Использование данного формата представления позволяет генерировать различные объекты в таких 3D программах как 3D studio max и использовать в данной программе

* 1. Описание трёхмерных афинных преобразований

Для трехмерного пространства любое аффинное преобразование может быть представлено последовательностью 3-х операций: сдвига, масштабирования, поворота.

Ниже приводятся уравнения и матрицы преобразований (вектор координат точки рассматривается как матрица с одной строкой и четырьмя столбцами)

* сдвиг точки на dx, dy, dz по координатным осям:

 ;

* масштабирование относительно начала координат с коэффициентами kx, ky, kz:

 ;

* поворот относительно осей x, y, z на угол :
  + ось x:

* ось y:

* ось z:

* 1. Описание алгоритмов произвольных преобразований камеры

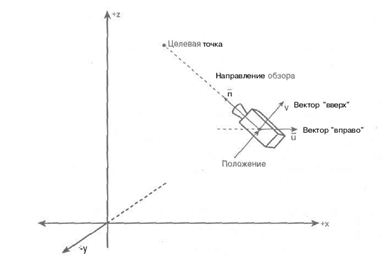
Для построения изображения на экранной плоскости используется модель представления при помощи камеры. Она отвечает за предоставление пользователю мощнейшего механизма движения среди объектов сцены.

Камера задается следующими параметрами:

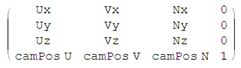
1. Углом обзора;
2. Фокусным расстоянием (от точки нахождения камеры до плоскости, на которую мы проецируем);
3. Ближними и дальними плоскостями, которые участвуют в отсечении по пирамиде видимости.

Для камеры были реализованы: поворот, смещение и панорамирование.

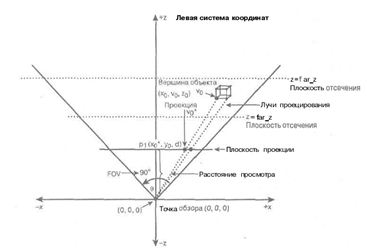
Камера расположена в определенной точке пространства, а также имеет целевую точку. На основании этих двух точек рассчитываются три вектора: вверх, вправо и вперед, которые однозначно определяют направление камеры и образуют оси системы координат камеры

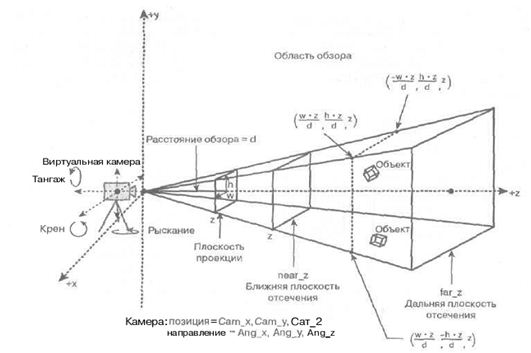


Для того чтобы получить координаты точки, представленной в мировых координатах, в координатах камеры необходимо эту точку умножить на матрицу камеры трансформации из МСК. Точку следует представить в однородных координатах. Матрица трансформации камеры приведена на рисунке. В этой матрице U – вектор «вправо», V – вектор «вверх», N – вектор «вперед», camPos – позиция камеры в МСК.



После получения координат объекта в системе координат камеры получаем аксонометрическую проекцию объекта на плоскости проекций камеры. На рисунке приведена упрощенная двумерная проекция плоскости обзора на плоскость xz. Примем угол обзора FOV равным 90 градусов, а расстояние от камеры до плоскости проекций d равным 1.



Перед нахождением перспективной проекции, нужно отсечь полигоны по пирамиде видимости

# 3. Технологическая часть

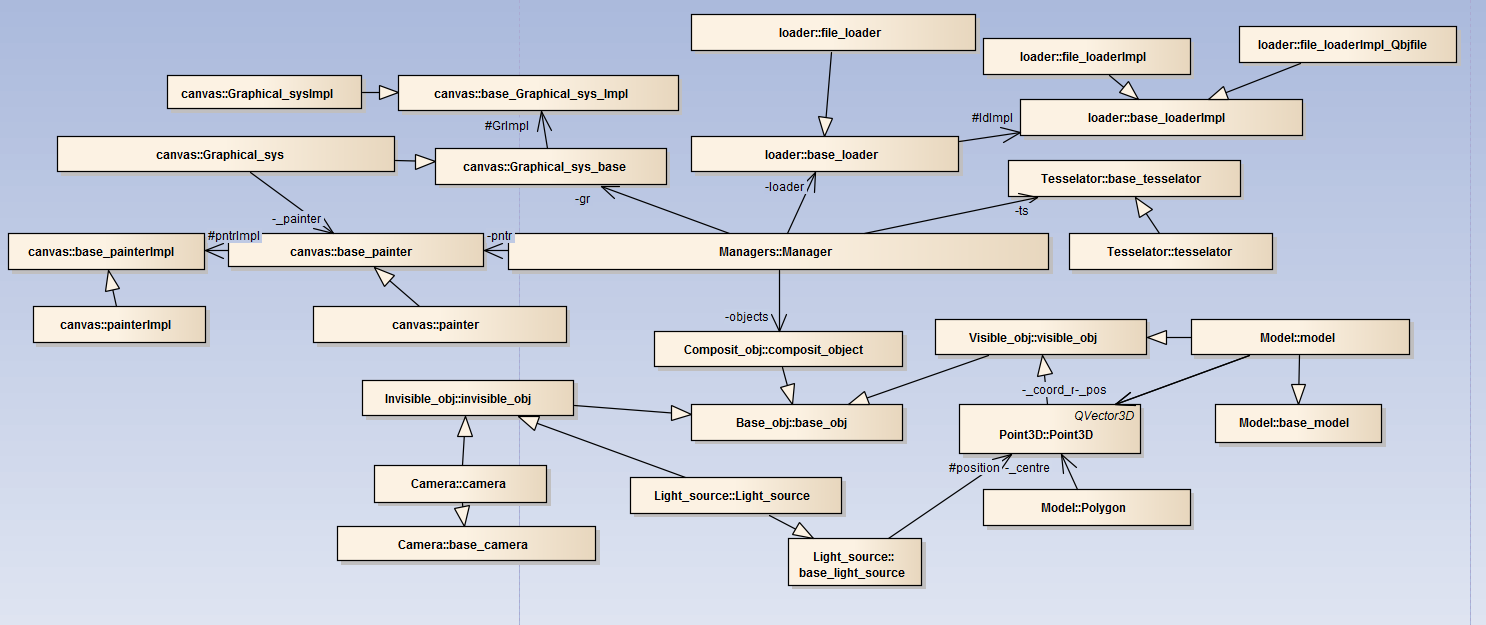
* 1. Выбор и обоснование языка и среды программирования

Для разработки данной программы применён язык C++ и среда Qt Creator по следующим причинам:

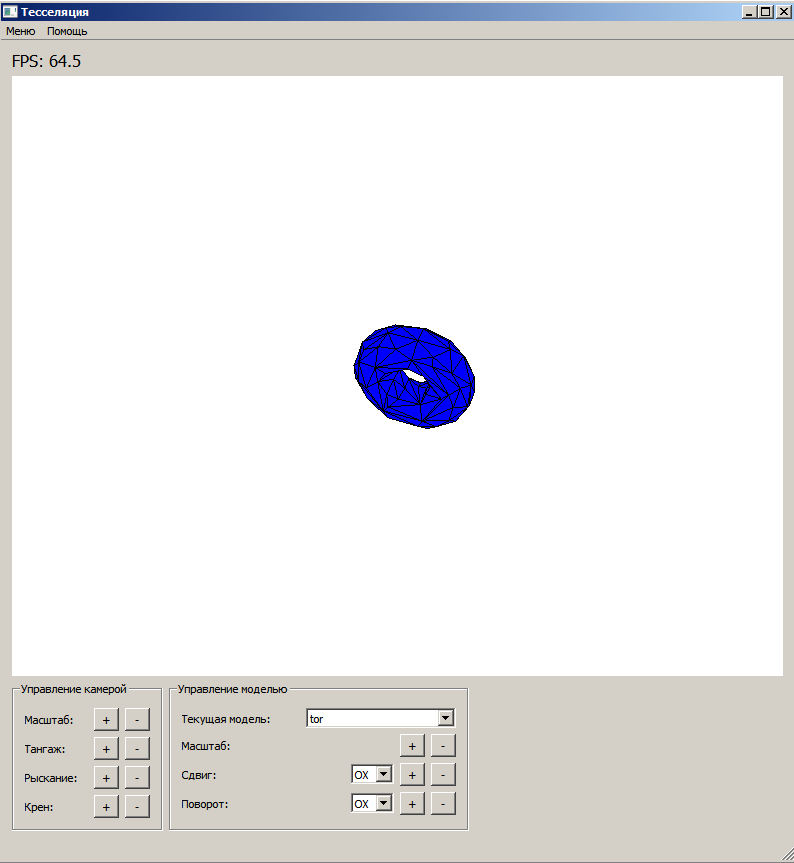
* широкие возможности реализации алгоритмов с высокой эффективностью;
* значительный набор стандартных классов и процедур; большое количество виджетов и их гибкость;
* кроссплатформенность;
* удобство отладки и редактирования программы;
* высокая скорость разработки и работы программы
* возможность вручную выделять и освобождать память, что обеспечивает более эффективное ее использование (например, в C# garbage collector включается только через определенные промежутки времени, поэтому после того, как данные станут не нужны, они будут занимать память очень долгое время).
* Возможность применения кроссплатформенной библиотеки для распараллеливания вычислений - OpenMP
  1. Описание структуры программы

В этой программе было решено использовать парадигму объектно-ориентированного подхода в связи со следующими его возможностями:

* Удобное написание кода
* Инкапсуляция переменных
* Механизм наследование классов
* Возможность локальной смены алгоритма без перестраивания логики выполнения всей программы

UML диаграмма классов:

* 1. Описание интерфейса программы



Это главное окно программы. Непосредственно в центре изображен тесселируемый объект. Строка с надписью «FPS» показывает частоту смены кадров в данный момент (frame per second – кадров в секунду, англ.)

Во вкладке «Меню» – выпадающий список с вкладкой для загрузки модели.

Во вкладке «Помощь» – выпадающий список со следующими вкладками:

* При нажатии на пункт меню «Управление» на экране появляется подсказка о способе управлении моделью в данной программе.
* При нажатии на пункт меню «О программе» на экране появляется подсказка о том, что можно сделать в этой программе, что собой представляет тесселяция, а также информация о ее создателе.

# Экспериментально-исследовательская часть

* 1. Характеристики оборудования
* Процессор: Intel Core i5 4210M 1.7GHz
* Оперативная память: 8GB
* Операционная система: Ubuntu 16.04
  1. Результаты исследования

Все 3 объекта выдали примерно одинаковую кривую частоты смены кадров в секунду. Что логично, так как алгоритм никаким образом не зависит от формы объекта, а только от числа полигонов. При максимальном отдалении FPS вырастает примерно в 5 раз для всех трех объектов.

В то же время дополнительные затраты памяти не так существенны – в худшем случае мы будем хранить 2 высоко полигональные модели вместо одной, т.е. ухудшение в 2 раза

Таким образом, данный алгоритм тесселяции дает большое преимущество в скорости работы, не сильно проигрывая по используемой памяти. Однако заметны сильные ухудшения скорости работы при большом числе полигонов, что не позволяет использовать его в реальном динамическом режиме.

# Заключение

Реализованная программа полностью соответствует техническому заданию, а также решает поставленные задачи, а именно:

* Осуществляет тесселяцию произвольного реалистичного объекта при условии наличия корректных входных данных
* Дает возможность оценить быстродействие алгоритма тесселяции путем вывода на экран информации о частоте отрисовки модели (FPS)

Эта работа в целом и отдельные алгоритмы из нее могут быть использованы повсеместно, где нужно отрисовывать на экране трехмерные реалистичные объекты при этом показывая неплохую производительность. Алгоритм позволяет получать промежуточные модели за достаточно быстрое время. Однако алгоритм нельзя использовать при динамической тесселяции. Так как время вычисления новых полигонов значительно повышается при их изменении в сторону увеличения.

# Список использованной литературы

1. *Роджерс Д., Адамс Дж.* Математические основы машинной графики. — М.: Мир, 2001.
2. Gerald E. Farin (2002). [Curves and Surfaces for CAGD: A Practical Guide](http://www.farinhansford.com/books/cagd/) (5th ed.). Morgan Kaufmann. [ISBN](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [978-1-55860-737-8](https://en.wikipedia.org/wiki/Special:BookSources/978-1-55860-737-8).
3. Abi-Chahla, Fedy. "Tessellation". Tom's Hardware. Retrieved 27 April 2013.
4. Tariq, Sara. ["D3D11 Tessellation"](http://developer.download.nvidia.com/presentations/2009/GDC/GDC09_D3D11Tessellation.pdf). Nvida. Retrieved 27 April 2013.
5. Скворцов А. В. Триангуляция Делоне и её применение. Томск: Изд-во Томского университета, 2002. 128 с. ISBN 5-7511-1501-5
6. Leach, G. (June 1992). "Improving Worst-Case Optimal Delaunay Triangulation Algorithms". CiteSeerX: 10.1.1.56.2323
7. [Foley, James](http://en.wikipedia.org/wiki/James_D._Foley); Feiner, Steven K.; [Hughes, John F.](http://en.wikipedia.org/w/index.php?title=John_f._Hughes_(Computer_Scientist)&action=edit&redlink=1) (1990). [*Computer Graphics: Principles and Practice*](http://en.wikipedia.org/wiki/Computer_Graphics:_Principles_and_Practice). Reading, MA, USA: [Addison-Wesley](http://en.wikipedia.org/wiki/Addison-Wesley). p. 1174. [ISBN](http://en.wikipedia.org/wiki/International_Standard_Book_Number) [0-201-12110-7](http://en.wikipedia.org/wiki/Special:BookSources/0-201-12110-7).