МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

01.03.02 Прикладная математика и информатика

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

БАКАЛАВРСКАЯ РАБОТА

на тему Разработка технологии хранения и анализа STL файлов

Студент К. А. Билько

Руководитель А. В. Очеповский

Допустить к защите
Заведующий кафедрой к.тех.н, доцент, А.В. Очеповский _____

«_____»_____20_____г.

Тольятти 2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав.кафедрой«Прикладная
математика и информатика»
_____А.В.Очеповский

«____»_____20__ г.

ЗАДАНИЕ
на выполнение бакалаврской работы

Студент Билько Константин Андреевич
1. Тема Разработка технологии хранения и анализа STL файлов
2. Срок сдачи студентом законченной выпускной квалификационной работы 24.06.2016
3. Исходные данные к выпускной квалификационной работе
- режим работы 24/7/365;
- использование языка высокого уровня;
- анализ трехмерных объектов;
- хранение трехмерных объектов.

4. Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов, разделов)

Введение
1. Технология быстрого прототипирования
   1.1 Технология печати и прототипирования данных
   1.2 Описание STL файла
   1.3 Требования к разрабатываемой системе
2. Анализ STL файлов
   2.1 Современные способы анализа STL
   2.2 Способы анализа формы при помощи топологии
   2.3 Эвристический анализ формы
3. Реализация программы для демонстрации работы алгоритмов
   3.1 Разработка архитектуры ПО
   3.2 Разработка базы данных
   3.3 Разработка программы

Заключение

Список использованной литературы

5. Ориентировочный перечень графического и иллюстративного материала Презентация, включающая блок-схемы работы приложения, графики, диаграммы, экранные формы, демонстрирующие работоспособность программного продукта

6. Дата выдачи задания « 11 » января 2016 г.

Руководитель выпускной квалификационной работы _____ А.В. Очеповский

Задание принял к исполнению _____ К. А. Билько

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав.кафедрой«Прикладная
математика и информатика»
_____А.В.Очеповский

«____»_____2016 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения бакалаврской работы

Студента Билько Константина Андреевича
по теме Разработка технологии хранения и анализа STL файлов

Наименование раздела работы	Плановый срок выполнения раздела	Фактический срок выполнения раздела	Отметка о выполнении	Подпись руководителя
Изучение способов хранения файлов	11.01.2016	11.01.2016	выполнено	
Изучение способов представление STL файлов	19.01.2016	19.01.2016	выполнено	
Изучение способов получения дополнительной информации из STL файлов	20.02.2016	20.02.2016	выполнено	
Разработка структуры программы и БД	8.03.2016	8.03.2016	выполнено	

Написание кода программы для хранения STL файлов	22.03.2016	22.03.2016	выполнено	
Написание кода анализа STL файлов	14.04.2016	14.04.2016	выполнено	
Подведение итогов, редактирование бакалаврской работы.	10.04.2016	10.04.2016	выполнено	
Создание презентационного материала	21.04.2016	21.04.2016	выполнено	
Проверка на наличие заимствований (плагиата) в системе antiplagiat.ru	24.05.2016	25.05.2016	выполнено	
Предварительная защита	30.05.2016-10.06.2016	31.05.2016	выполнено	
Сдача на кафедру отзыва научного руководителя и ознакомление с ним	20.06.2016	20.06.2016	выполнено	
Сдача на кафедру комплекта документов для защиты	24.06.2016	24.06.2016	выполнено	
Защита ВКР	27-29.06.2016	29.06.2016	выполнено	

Руководитель выпускной квалификационной работы _____ А. В. Очеповский

Задание принял к исполнению _____ К. А. Билько

Аннотация

Темой данной выпускной квалификационной работы является «Разработка способов хранения и анализа STL файлов»**.**

Работа выполнена студентом Тольяттинского Государственного Университета, института математики, физики и информационных технологий, группы ПМИб-1201, Билько Константином Андреевичем.

Объект исследования является процесс хранения и анализа трехмерных моделей.

Предмет исследования являются алгоритмы анализа формы трехмерный объектов хранящихся в STL файлах.

Цель исследования является разработка алгоритма анализа формы трехмерных объектов.

Для достижения поставленной цели требуется выполнить следующие **задачи:**

- изучить основные современные способы построения трехмерных моделей;
- разработать способ хранения трехмерных моделей;
- изучить методы анализа трехмерных объектов;
- создать демонстрационное приложение.

Отчет состоит из введения, трех глав и заключения.

Во введении описываются цели, задачи и актуальность данной работы.

В первой главе рассказывается о технологии быстрого прототипирования файлов. Описывается процесс создания твердотельных объектов. Разбирается структура STL файла и его особенности. Так же задаются требования к программному продукту.

Во второй главе описываются способы анализа STL файла. В первой части рассказывается о современных способах анализа STL файлов. В второй части рассказывается о топологии. В третей части рассказывается о эвристическом алгоритме анализа формы объекта, который содержится в STL файле.

В третьей главе описан программный продукт, демонстрирующий работу алгоритмов описанных во второй главе.

Выпускная квалификационная работа представлена на 40 страницах, включает 34 иллюстрации, 2 формулы, 1 приложение, список используемой литературы содержит 21 источник.

Оглавление

Введение ... 3

Глава 1 Технология быстрого прототипирования ... 6

1.1 Технология печати и прототипирования данных ... 6

1.2 Описание STL файла ... 8

1.3 Требования к разрабатываемой системе ... 11

Глава 2 Анализ STL файлов ... 13

2.1 Современные способы анализа STL ... 13

2.2 Способы анализа формы при помощи топологии ... 16

2.3 Эвристический анализ формы ... 19

Глава 3 Реализация программы для демонстрации работы алгоритмов ... 28

3.1 Разработка архитектуры ПО ... 28

3.2 Разработка базы данных ... 28

3.3 Разработка программы ... 30

Заключение ... 37

Список используемой литературы ... 39

Приложение А Листинги кода реализации файлов ... 41

Введение

Рынок быстрого протипирования или 3D-печати, это быстро развивающийся сегмент экономики. Интерес к данной отрасли экономики растет с каждым годом, так как дает новые возможности в бизнесе. На рисунке 1 показана тенденция развития рынка [16].

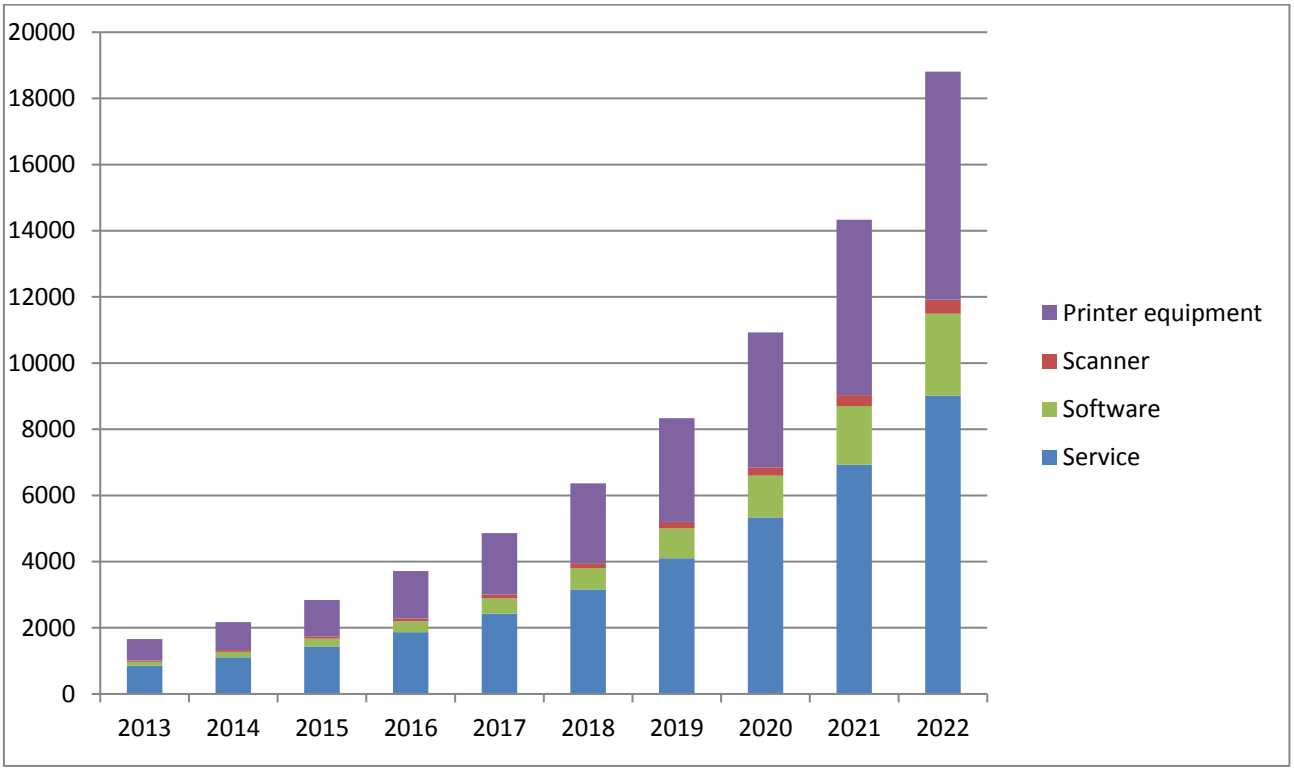

Рисунок 1 - Тенденция развития рынка 3D-печати

Как видно на рисунке 1, этот рынок быстро развивается и с каждым готов увеличивается в среднем на 30%. К 2020 году рынок может вырасти в 2-3 раза.

Основная часть рынка принадлежит продаже принтеров и их обслуживанию. Эти два показателя влияют на развитие рынка приложений и сервиса.

Помимо печати твёрдотельных объектов, фирмы так же зачастую выполняют работу создания трехмерным моделей для будущей печати. Создание таких моделей ведётся при помощи специальных программ или сканера, который позволяет сразу получить трехмерную модель. Но сканер работает только в случае, когда уже имеется прототип. Фактически он используется только для создания копии, поэтому рынок сканеров развивается медленнее, чем остальные.

С увеличением рынка производства трехмерных моделей появилась необходимость хранить ранее созданных модели в базах данных. Для разработки новых трехмерных моделей иногда используют старые модели. Но перед этим их надо найти. Со временем в фирме появляется столько трехмерных моделей, что найти нужную модель среди них достаточно сложно.

Так же помимо увеличения количества файлов, хранящихся в базе данных, на рынке 3D-печати существует проблема повторного создания трехмерной модели. Это случается тогда, когда среди старых файлов, разработчик не находит нужный и начинается разработку заново, хотя такая модель уже была.

Все эти проблемы связаны с тем, что в настоящее время анализ трехмерных моделей фактически не производится. В основном анализ производиться только для выявления ошибка и улучшения качества производимых твердотельных объектов. В свою очередь трехмерные модели содержать в себе форму, по которой человек может понять, была ли уже создана такая модель. Но анализ формы трехмерных моделей не производится, так как ранее, когда рынок не был ещё так развит, это не требовалось.

Актуальность проблемы заключается в то, что ожидаемый рост рынка сервисов и программ, для 3D-печати обуславливает исследование новых информационных технологий, для работы с файлами трехмерный моделей.

Новизна заключается в создании алгоритмов, позволяющих различать трехмерные объекты по форме.

Объект исследования является процесс хранения и анализа трехмерных моделей.

Предмет исследования являются алгоритмы анализа формы трехмерный объектов хранящихся в STL файлах.

Цель исследования является разработка алгоритма анализа формы трехмерных объектов.

Для достижения поставленной цели, требуется выполнить следующие **задачи:**

- изучить основные современные способы построения трехмерных моделей;
- разработать способ хранения трехмерных моделей;
- изучить методы анализа трехмерных объектов;
- создать демонстрационное приложение.

Отчет состоит из введения, трех глав и заключения.

Во введении описываются цели, задачи и актуальность данной работы.

В первой главе рассказывается о технологии быстрого прототипирования файлов. Описывается процесс создания твердотельных объектов. Разбирается структура STL файла и его особенности. Так же задаются требования к программному продукту.

Во второй главе описываются способы анализа STL файла. В первой части рассказывается о современных способах анализа STL файлов. Во второй части рассказывается о топологии. В третей части рассказывается о эвристическом алгоритме анализа формы объекта, который содержится в STL файле.

В третьей главе описан программный продукт, демонстрирующий работу алгоритмов описанных во второй главе.

Глава 1 Технология быстрого прототипирования

1.1 Технология печати и прототипирования данных

Прототипирование начинается с создания трехмерной модели. При создании используется полигонная сетка. На сегодняшний день это самый распространённый способ создания трехмерный моделей.

После созданная модель отправляется на печать.

В 1986 года Чарльзом В. Халлом, запатентовавшим метод и аппарат для производства твердых физических объектов за счет последовательного наслоения фотополимерного материала. Данный процесс получил термин стереолитография [17,18].

Стереолитография – технология аддитивного производства моделей, прототипов и готовых изделий из жидких фотополимерных смол. Отвердевание смолы происходит за счет облучения ультрафиолетовым лазером или другим схожим источником энергии [1].

Данный процесс послужил началу создания принтеров, способных печатать трехмерные модели. Данные принтеры позволяют быстро создать прототип макета. Благодаря тому, что объект воспроизводиться машиной, избегаются ошибки, которые присутствуют при ручном создании прототипа.

Несмотря на то, что существует множество технологий и материалов для создания прототипа, все они основаны на одном и том же методе.

Сегодня, при помощи данного процесса, создаются машины, протезы, печатаются вены и даже человеческие ткани. Данных процесс дал большой толчок к развитию архитектуры, биологии, машиностроения, авиастроения и многого другого.

Как видно на рисунке 1.1, принтер слой за слоем наносит вещество, из которого изготавливается прототип, на рабочую поверхность.

Как видно на рисунке 1.2, после нанесения первого слоя, второй слой наноситься на первый.

Заполнение контура внутри может сильно отличаться, в зависимости от метода и способа построения объекта.

Данный процесс продолжается до тех пор, пока не будет воспроизведена вся модель.

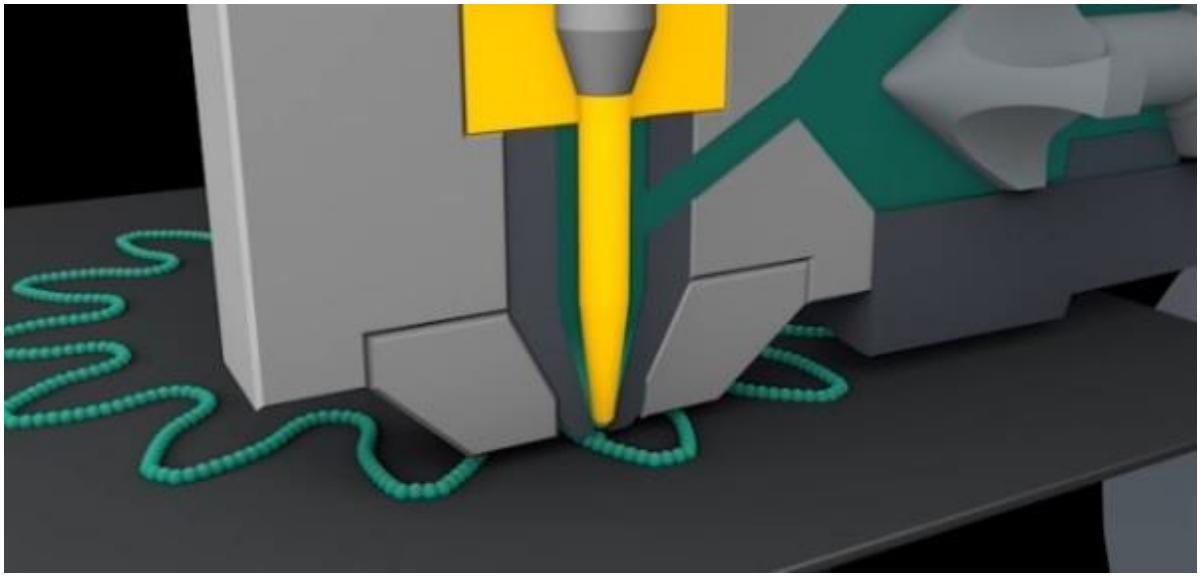

Рисунок 1.1 - Печать прототипа на 3D-принтере

Такой метод печати накладывает свои ограничения на способ представления трехмерных моделей, поскольку помимо отображения трехмерной модели на экране компьютера, требуется ещё и преобразование модели в слои, для 3D-принтера.

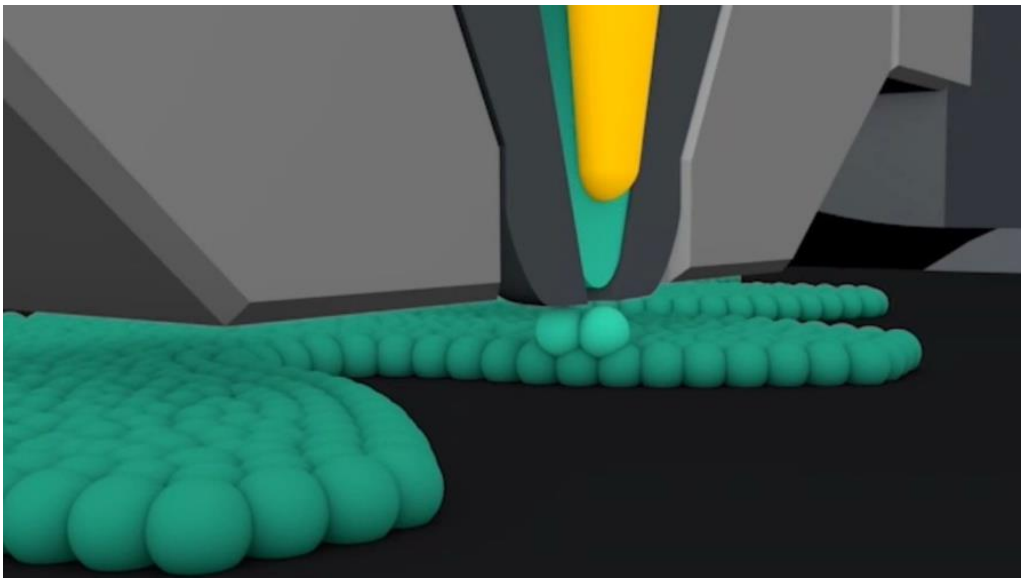

Рисунок 1.2 - Нанесение второго слоя вещества

Процессу «нарезки» трехмерной модели на слои уделяется очень много внимания, так как от этого зависит скорость построения модели, ошибки при нарезке, прочность модели и другие свойства. Данный параметр очень сильно

влияет на конечную модель, но при этом сам зависит от трехмерной модели. Вследствие чего, при разработке любой трехмерной модели уделяется много внимания тому, как она строиться, чтобы добиться максимального качества воспроизведения модели [19].

Перед разработчиками стояла задача, разработать алгоритм для построения трехмерной модели, а так же последующим «нарезанием» её на слои, с учетом всех требований.

1.2 Описание STL файла

Первый алгоритм был разработан компанией Albert Consulting group по заказу от компании 3D Systems. Данной компанией был разработан формат STL и спецификации к нему. В 1987 году компания 3D Systems открыто опубликовала STL-формат [21].

В STL-формате модель представляется в виде последовательности треугольников (фасетов). Каждый фасет описывается набором координат, для трех вершин, и нормальным вектором. Так же современные производители добавляют значение цвета в формате RGB. На рисунке 1.3 показан код фасета.

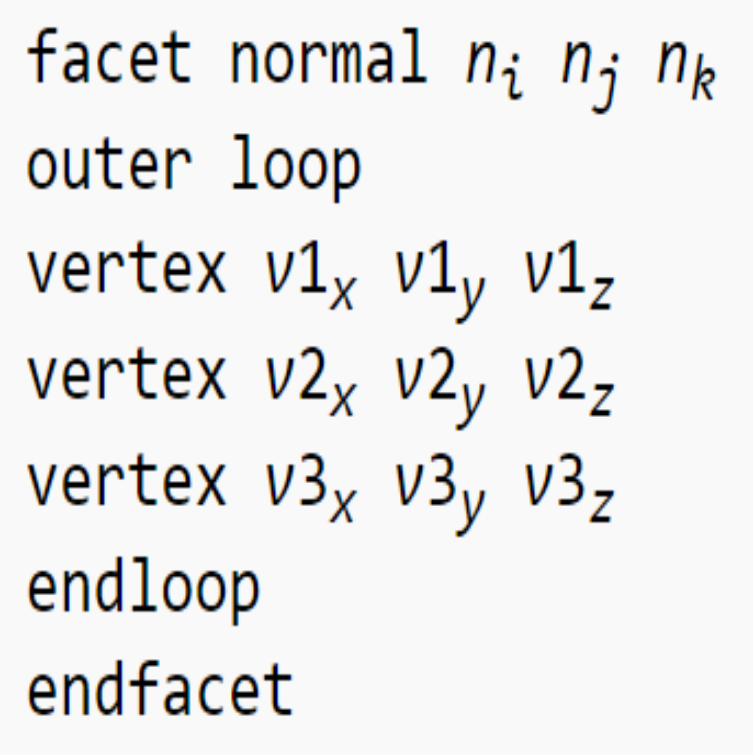

Рисунок 1.3 - Код фасета

Сохранение STL файла происходит в бинарном формате или в формате ASCII. Поэтому некоторые STL файлы в программах открываются не корректно.

На рисунке 1.4 изображен фасет в трехмерном пространстве. Направление нормали определяется по правилу буравчика.

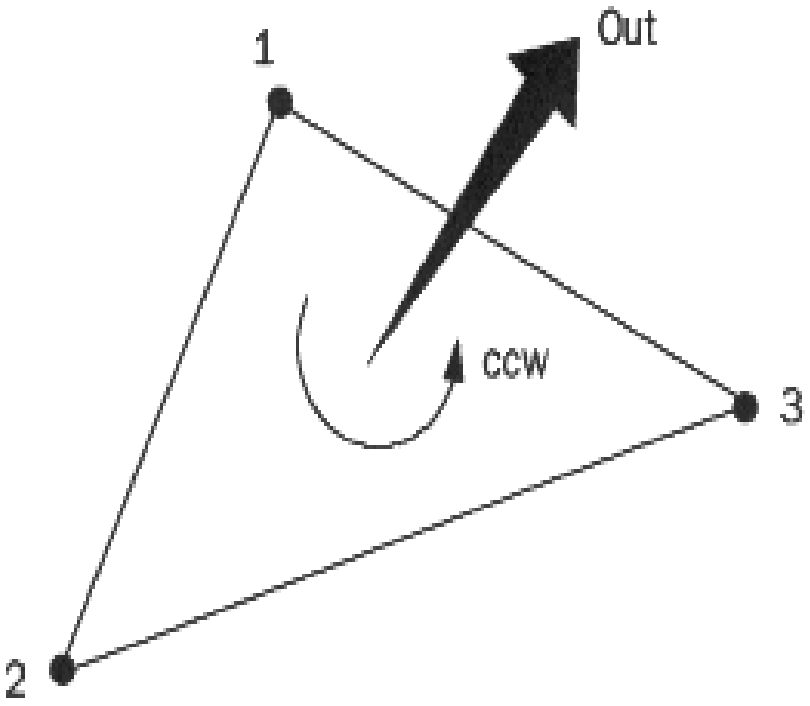

Рисунок 1.4 - Фасет в трехмерном пространстве

Данный вид представления моделей напоминает нам полигонное представление трехмерной модели. Отличие состоит в использовании нормалей. Нормали нам необходимы для определения внешней и внутренней стороны трехмерной модели. Направление нормали определяется по правилу буравчика.

STL-формат стал стандартом, для передачи данные в устройство быстрого прототипирования, за счет удобства представления трехмерной модели и удобного алгоритма нарезки. На рисунке 1.5 показана STL-модель бокала. На ней отчетливо видно, как модель строится их фасетов.

Для создания STL моделей, разработчики пользуются различными программами средствами. Это ускоряет производство трехмерных моделей.

В настоящее время при разработке моделей, разработчики стараются увеличить количество фасетов, чтобы сделать модель более гладкой [10].

Рисунок 1.5 - STL модель бокала

Как видно на рисунке 1.6, чем более детализирована модель, тем более качественно получается готовое изделие. Увеличение количества фасетов ведёт не только к улучшению качества получаемой модели, но так же значительно увеличивает время создания твердотельной модели. В следствии чего, разработчиками постоянно ищется «золотая» середина, между качеством и скорость [20].

Так же на рисунке 1.6 видно, что есть предел качества детализации модели. Он зависит от 3D-принтера, на котором воспроизводиться модель. В зависимости от минимальной толщины слоя «нарезки» меняется и качество готового изделия.

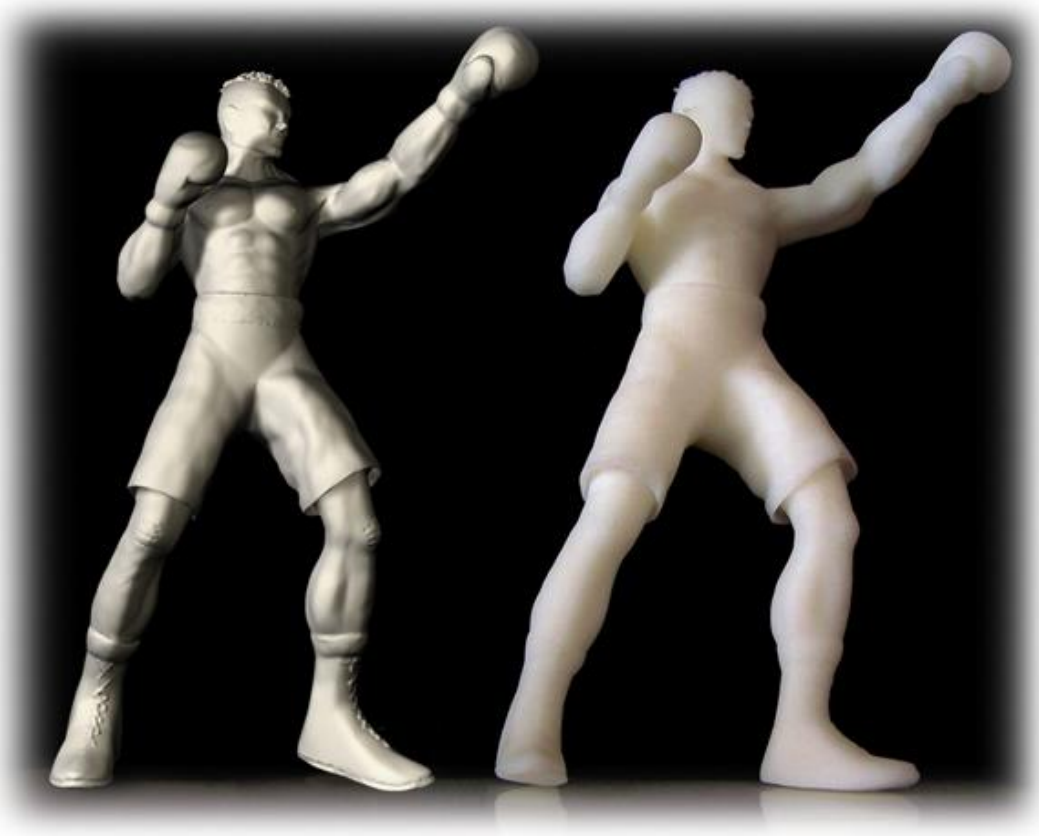

Рисунок 1.6 - 3D-модель и готовое изделие

Именно поэтому разработчики STL-файлов зачастую разрабатывают модели под определённые характеристики принтера. То есть под определённый минимальный слой «нарезки». Это делается для того, чтобы убрать излишнее время сборки и ошибки, которые могут быть в более детализированной модели.

1.3 Требования к разрабатываемой системе

STL-файлы описывают определенные твердотельные объекты, которые используются человеком в повседневной деятельности, поэтому их можно классифицировать по группе принадлежности объекта. В качестве классификации лучше всего подходит название объекта, который описывает модель. К примеру, у кольца может быть несколько моделей, но группа классификации этих объектов будет одна.

Так же все объекты имеют геометрические показатели, такие как вес, размер, объем. Файл STL не содержит этих сведений. Так же эти данные невозможно получить напрямую из файла. Любой трёхмерный объект можно масштабировать, а, следовательно, все его показатели будут изменяться.

Поэтому эти данные должен вносить оператор, который разрабатывает модель. Эти данные могут потребоваться для перерасчета показателей объекта, если другой человек будет масштабировать объект.

Разработкой STL файлов занимаются несколько человек. В результате может возникнуть ситуация, при которой разные STL объекты будут иметь одинаковые имена.

Помимо данных, которые можно получить от разработчиков STL файлов, программа должна анализировать структуру файлов, чтобы получать дополнительную информацию.

Все эти данные накладывают определённые требования на разрабатываемую систему. В программе должны быть реализованы следующие функции:

- сохранение STL файлов и информации о них в базе данных;
- просмотр трехмерных объектов, которые содержаться в STL файлах;
- отображения данные, которые хранятся в базе данных и полученные в ходе анализа;
- добавление новых файлов в базу данных.

Глава 2 Анализ STL файлов

2.1 Современные способы анализа STL

В настоящее время основное направление анализа STL файлов направлено на поиск ошибок в файле [18].

Наиболее распространённые ошибки:

- некорректный нормальный вектор мозаичной поверхностей с большой кривизной может привести к неправильной ориентации граней, которые в свою очередь, означает не подчинение правилу ориентации граней;
- пробелы (недостающие грани);
- вырождение граней;
- перекрытие граней.

Мозаичная поверхность с большой кривизной может привести к ошибкам при пересечениях между такими поверхностями, оставляя зазоры или отверстия вдоль краев модели детали, как показано жирными линиями на рисунке 2.1.

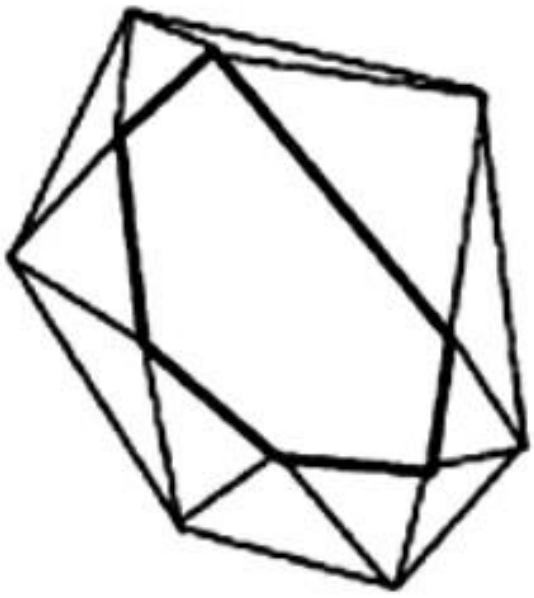

Рисунок 2.1 - Пример отсутствия фасетов

Геометрическое вырождение граней происходит тогда, когда все края фасета являются коллинеарными, даже если все его вершины различны. Это

может быть вызвано сшивание алгоритмов, которые пытаются избежать пропусков, как показано на рисунке 2.2.

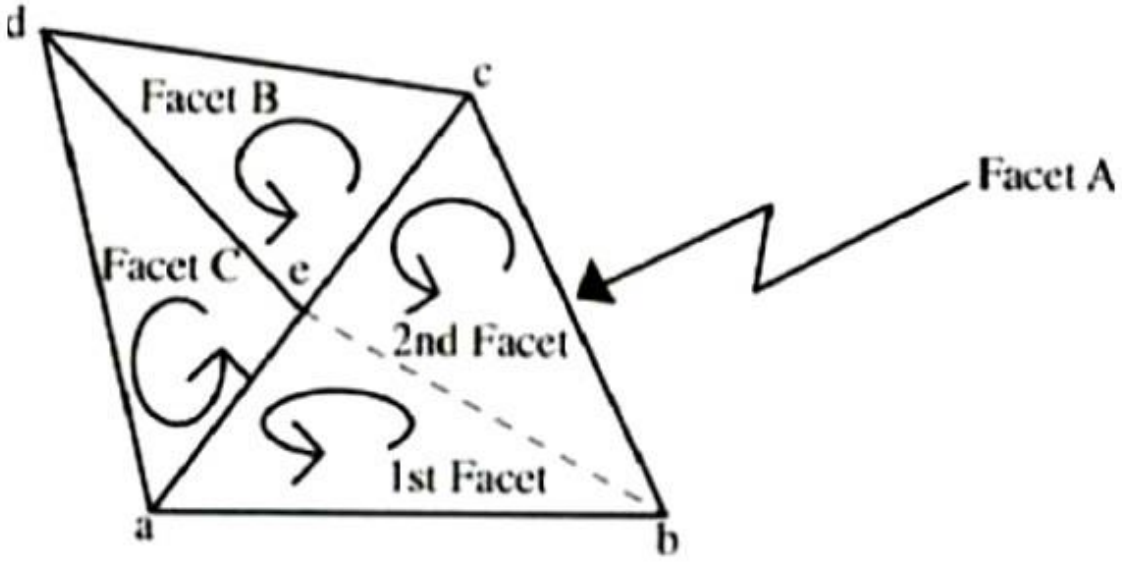

Рисунок 2.2 - Деградация граней

Перекрытия граней могут быть получены в результате численных ошибок округления во время тесселяции. Вершины представлены в 3-мерном пространстве в качестве чисел с плавающей точкой, а не целыми числами. Таким образом, целочисленное округление может привести к грани, чтобы перекрывать, если допуски установлены слишком свободно, как показано на рисунке 2.3.

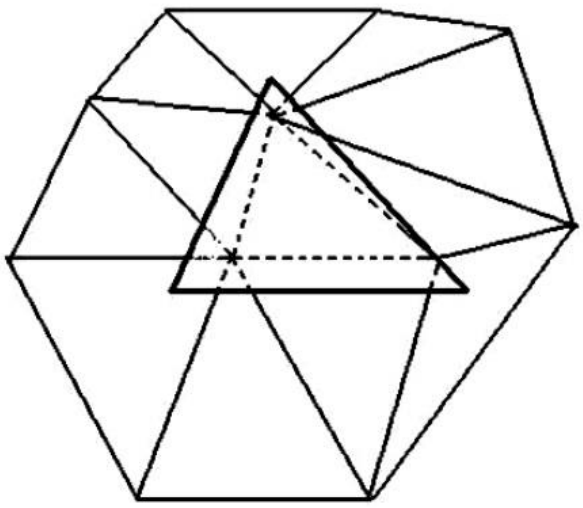

Рисунок 2.3 - Пример пересекающихся граней

Все эти ошибки могут привести к неправильной печати твердотельной модели. На рисунке 2.4 показано, как все эти ошибки могут повлиять на конечный продукт. Слева показан STL файл, а справа то, что будет печатать принтер.

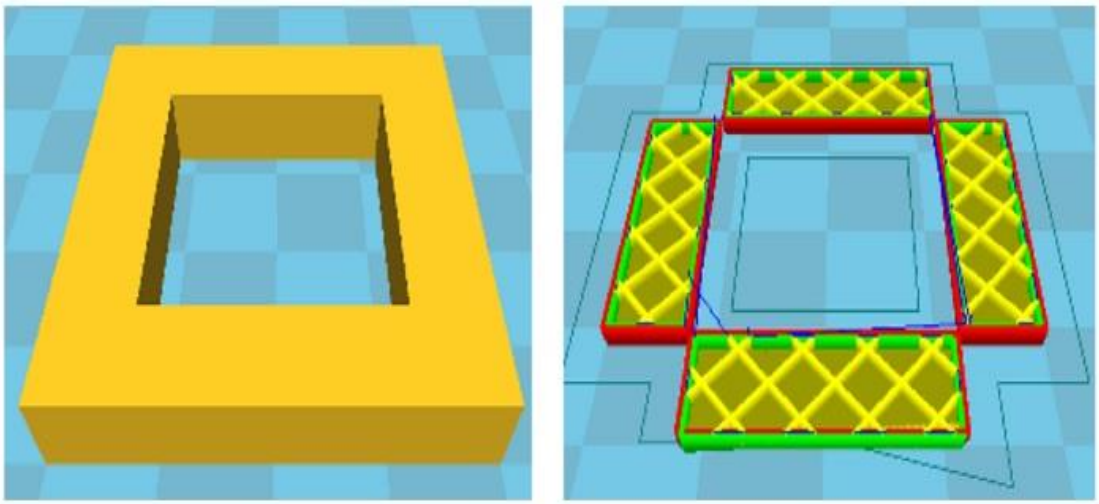

Рисунок 2.4 - Результат печати неправильного STL файла

Поэтому преждевременное их выявление очень важно. Сейчас существует множество методов и способов поиска ошибок в STL файла.

Ошибки, которые присутствуют в STL файле, связаны с его архитектурой. Ученые уже давно предлагают изменить структуру файла, чтобы избавиться от них. Но до сей пары, не было принято ни одной правке. Это связано с тем, что формат файла быстро стал популярным и поэтому любое изменение затронет множество файлов. Так же если в результате будут внесены неправильные изменения, то компании, занимающиеся 3D-печатью, понесут больше убытки.

Так же правки в STL файл не были внесены, поскольку нет единого мнения, как лучше всего модифицировать его структуру, чтобы избавиться от ошибок, но в тоже время оставить текущую гибкость системы. Поэтому сейчас существует много программ и методов, для быстрого обнаружения и устранения ошибок в файлах.

2.2 Способы анализа формы при помощи топологии

Основная проблема анализа формы трехмерных объектов заключается в возможности их масштабирования. Масштабирование трехмерного объекта может, проходит как в редакторе, так и при печати твердотельной модели. Трехмерные объекты строятся их фасетов, а не их формул. Анализ можно построить только на анализе фасетов, рёбер и точек [2].

Топология – это раздел математики, изучающий свойства объекта, которые не изменяются при непрерывных деформациях [3].

Под деформацией понимается растяжение, сжатие, изгибание трехмерного объекта без разрывов, или склеиваний. То есть мы можем изменять положение точек фасета, но не можем их объединять или удалять грани [4,5].

Всё трехмерные объекты имеют следующие топологические свойства:

- эйлерова характеристика;
- род поверхности.

Все топологические свойства основаны на анализе вершин, ребёр, граней. Эйлерова характеристика определяется по формуле 1.

$$V - E + F = 2 \quad (1)$$

где V – число вершин;

E – число ребёр;

F – число граней.

Если число Эйлера равно 2, то такие многогранники называются простыми, в противном случае не простыми. Данная характеристика не несёт для нас достаточно информации, так как она позволяет разделить объекты только на две группы [8, 9].

На рисунке 2.5 представлены две фигуры. Число Эйдера для левой фигуры, равно 2, а для правой 0 [11].

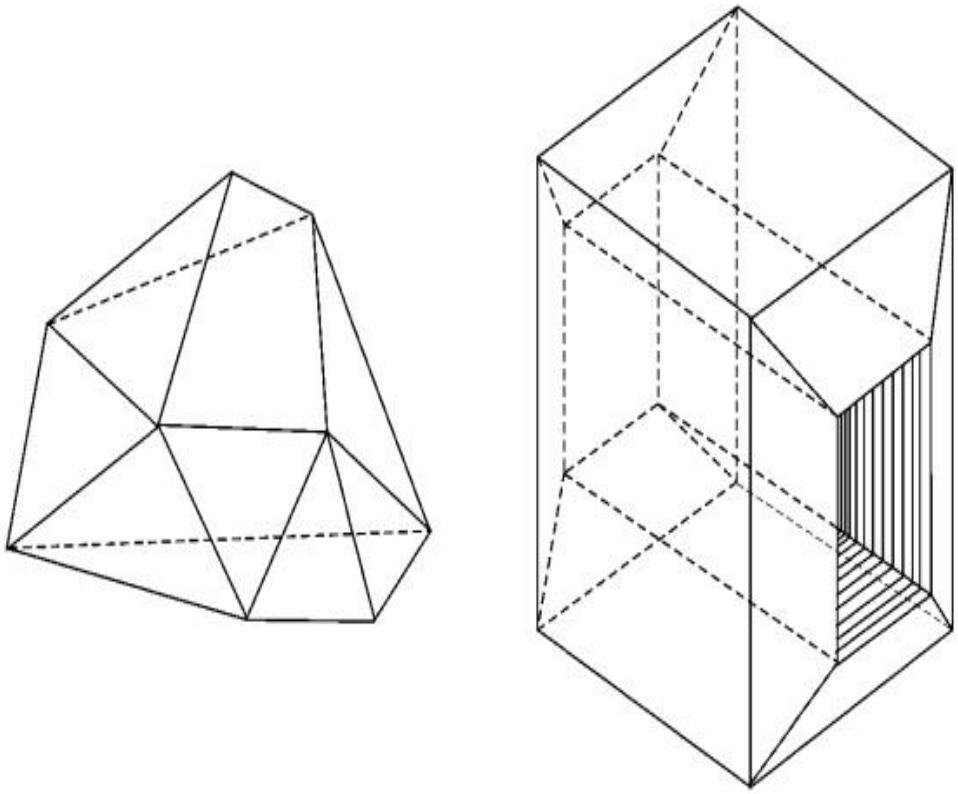

Рисунок 2.5 - Простой и не простой трехмерный объект

Род поверхности рассчитывается по формуле 2

$$V - E + F = 2 - 2 * p \quad (2)$$

где V – число вершин;

E – число ребёр;

F – число граней;

p – род поверхности.

Согласно топологическим преобразованиям тор, нельзя преобразовать в сферу [12].

Чтобы тор преобразовать в сферу её необходимо разрезать, как показано на рисунке 2.6. Таким же способом сферу можно преобразовать в тор. Другими словами после разреза тора, вся его поверхность становиться связанной. В свою очередь разрез на сфере, не приведёт ни к каким изменениям [7].

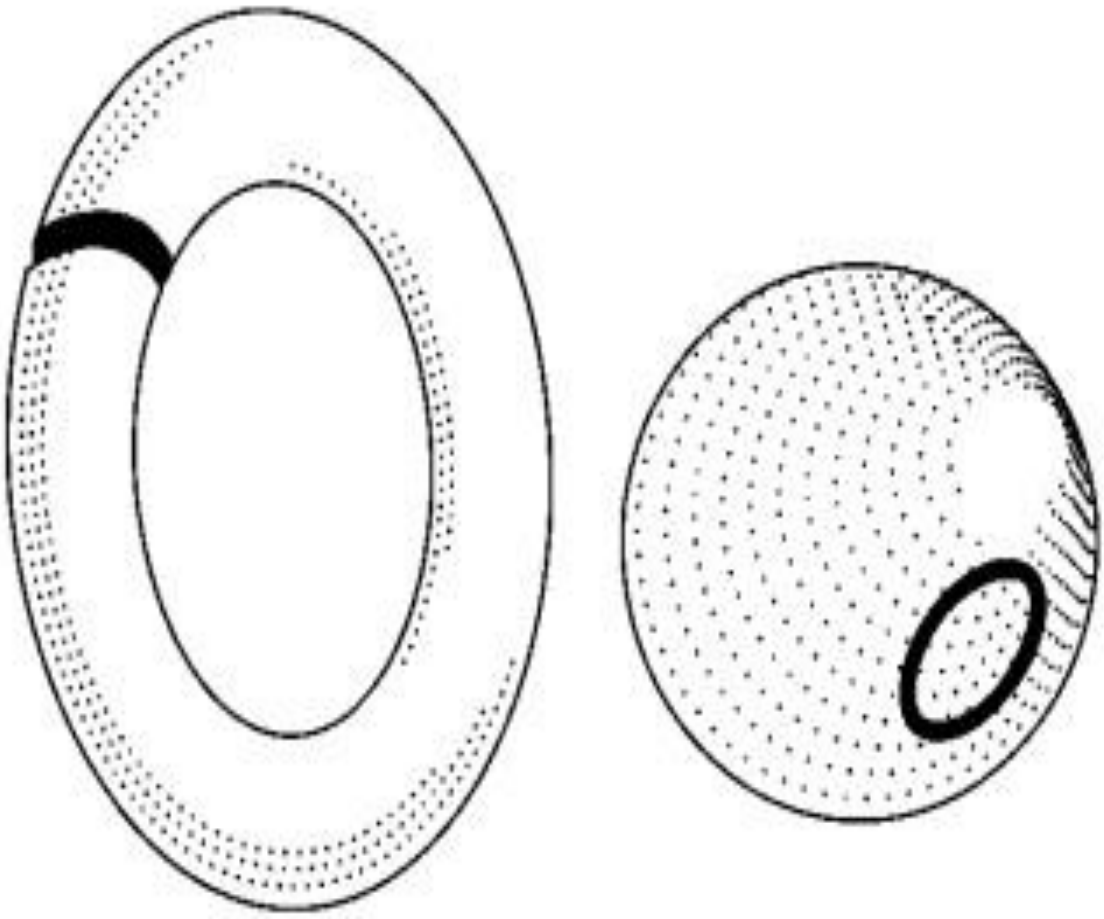

Рисунок 2.6 - Преобразование разрезанного тора в сферу

Рассмотрим поверхность с двумя дырками, изображенную на рисунке 2.7. На этой поверхности можно провести сразу две замкнутые кривые A и B, которые не разделят поверхность на части. Тор, напротив, при проведении двух таких кривых непременно разделиться на части. Конечно, фигура, изображенная на рисунке 2.7, может быть разрезана иначе, что тоже даст приведение её к простой. [6].

Род поверхности показывает минимальное количество разрезов, которое необходимо для приведения поверхности к сфере. Род сферы равен 0, род тора – 1, род поверхности на рисунке 2.7 - 2. Таким образом, род поверхности показывает, к какому классу относится поверхность [14, 15, 16].

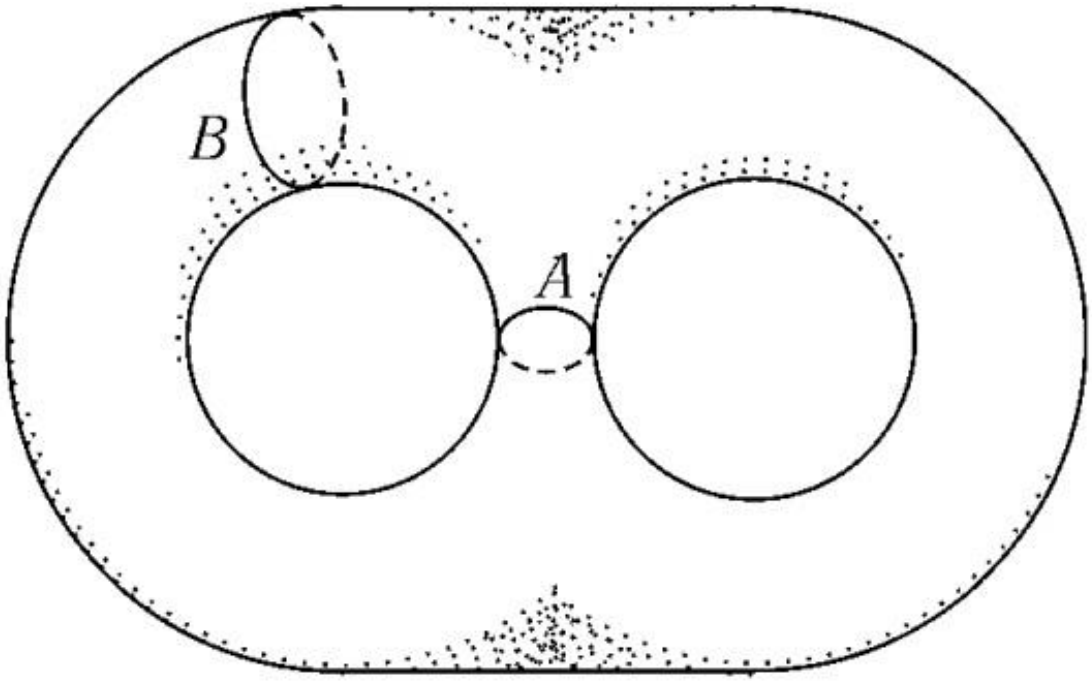

Рисунок 2.7 - Трехмерный объект с двумя "дырками"

Таким образом, род поверхности позволяет нам разделить объекты на несколько групп, в отличие от эйлеровой характеристики. Несмотря на это, количество объектов, которые могут иметь одинаковый род поверхности, достаточно много.

2.3 Эвристический анализ формы

Свойства, полученные с помощью топологии, не дают нам ни каких данные о форме объекта. Поэтому нам необходимо придумать свой способ анализа формы объектов.

Анализ будет строиться на основе угла между фасетами. Рассчитывая угол между фасетами, мы сможем построить трехмерную таблицу переходов. Как показано на рисунке 2.8, угол между 1 и 2 фасетом равен 180. Следовательно, они находится в одной плоскости. Угол между 1 и 3 фасетом равен 90 градусов. Следовательно, они лежать в разных плоскостях.

Пример трехмерной таблицы для квадрата показан на рисунке 2.9.

В зависимости от начального фасета меняется и таблица построения. После того, как мы высчитали смежные фасеты с фасетом 1, мы переходим к

следующему смежному фасету. Так продолжается до тех пор, пока мы не просчитаем все фасеты.

Надо отметить, что фасеты, которые уже были смежными, не могут снова участвовать в расчетах. Поэтому каждый фасет может быть «основным» и смежным только один раз. Так же нам надо запоминать, к какой части таблицы относиться выбранный фасет, чтобы правильно заполнять таблицу.

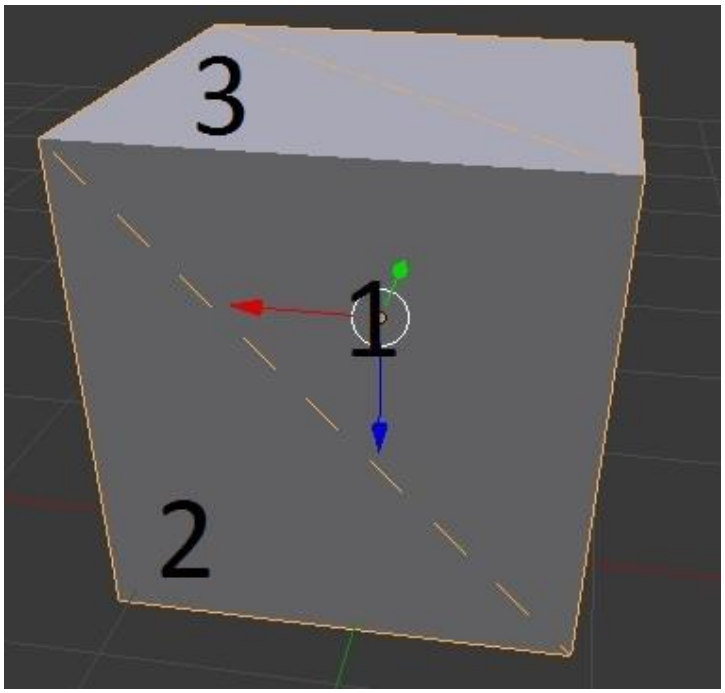

Рисунок 2.8 - STL квадрата с метками обхода

Основная проблема расчета угла между фасетами в том, что он не показывает, к какой стороне трёхмерной таблицы относится фасет.

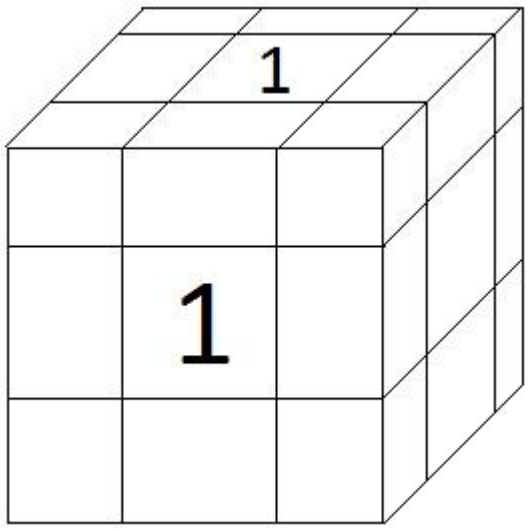

Рисунок 2.9 - Трехмерная таблица для квадрата

Рассмотрим проблему сравнения углов фасетов на примере тора, который изображен на рисунке 2.10. Если начальный фасет попадёт на внутренний круг тора, то он может считаться нижним.

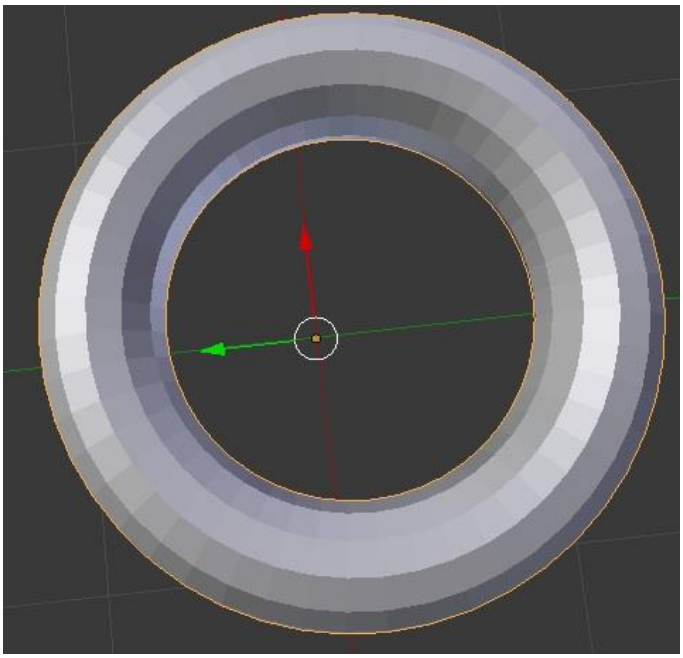

Рисунок 2.10 – Трёхмерная модель тора

Как показано на рисунке 2.12, угол между фасетами равен углу, между нормалями. В отличие от фасетов, их нормали показывают, с какой стороны расположена внутренняя и внешняя сторона фасета. Чтобы определить к какой части таблицы относиться нормаль, будем сравнивать их с нормалями трехмерной таблицы.

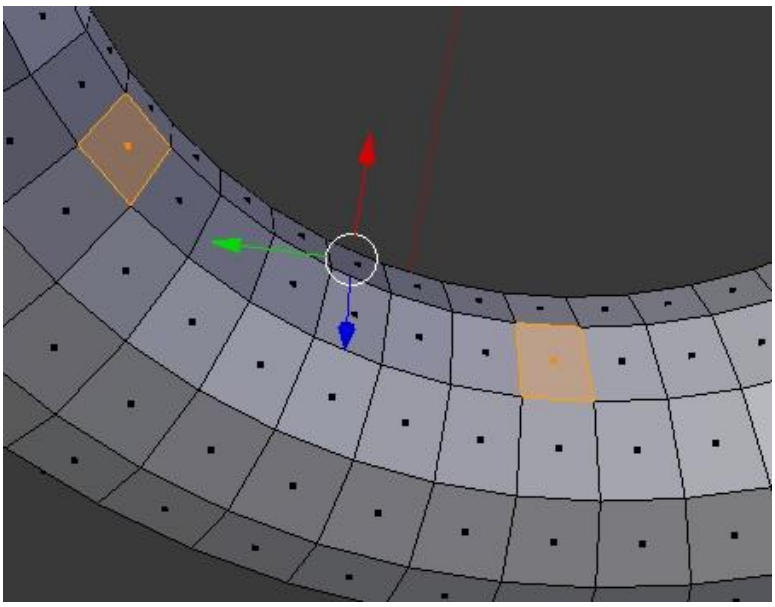

Рисунок 2.11 - Сравнение фасетов

На рисунке 2.13 показана трёхмерная модель нормалей трёхмерной таблицы. Благодаря этому мы можем точно определить, к какой части таблицы относиться выбранный фасет.

Нормаль, к которой относиться нормаль фасета, выбирается из значения меньшего угла между ними. Таким образом, у каждой нормали трехмерной таблицы есть свой радиус, в который попадают нормали фасетов. Данный радиус может меняться в зависимости от размера трехмерной таблицы. В нашем случае угол, между нормалями равен 45 градусов. В окружности у нас 8 нормалей. Следовательно, область захвата каждой нормали равна 40 градусов. Если мы увеличим размерность нашей таблицы до 4, то радиус захвата нормалей будет равен 30градусам. Увеличение размера трехмерной таблицу влечет к увеличению четкости получаемых данных. Но в тоже время если мы будем продолжать увеличивать размерность таблицы, то в ней отпадет вся необходимость, поскольку в таком случае каждое значение в таблице будет равно 0 или 1.

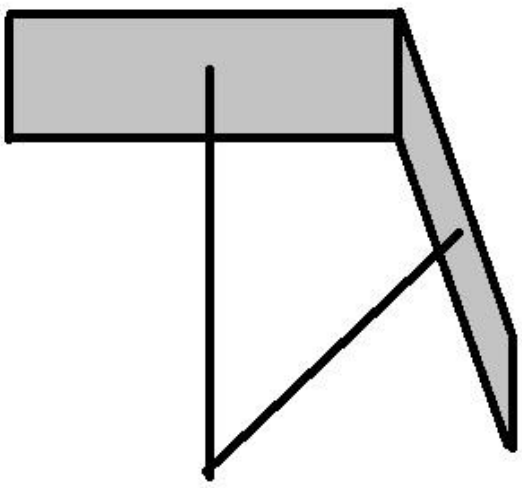

Рисунок 2.12 - Равенство углов фасетов и нормалей

При увеличении четкости STL модели увеличивается количество фасетов. Как показано на рисунке 2.14, один объект может иметь разное количество фасетов. Поэтому мы не будем учитывать смежные фасеты, лежащие в одной плоскости. Это позволит нам просчитать «менее четкую» форму объекта. Но такой подход накладывает свои ограничения.

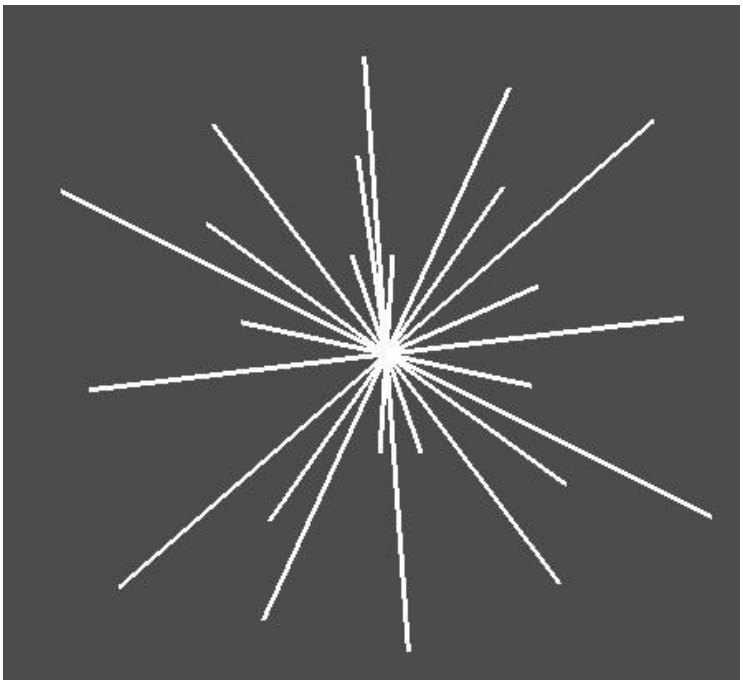

Рисунок 2.13 - Трёхмерная модель таблицы

Если мы не будем учитывать фасеты, лежащие в одной плоскости, то нам необходимо составить такой метод обхода всех фасетов, который учитывал бы эту характеристику. Если мы будем просто обходить все фасеты и рассчитывать разность нормалей, то в нашей таблице будет ошибка.

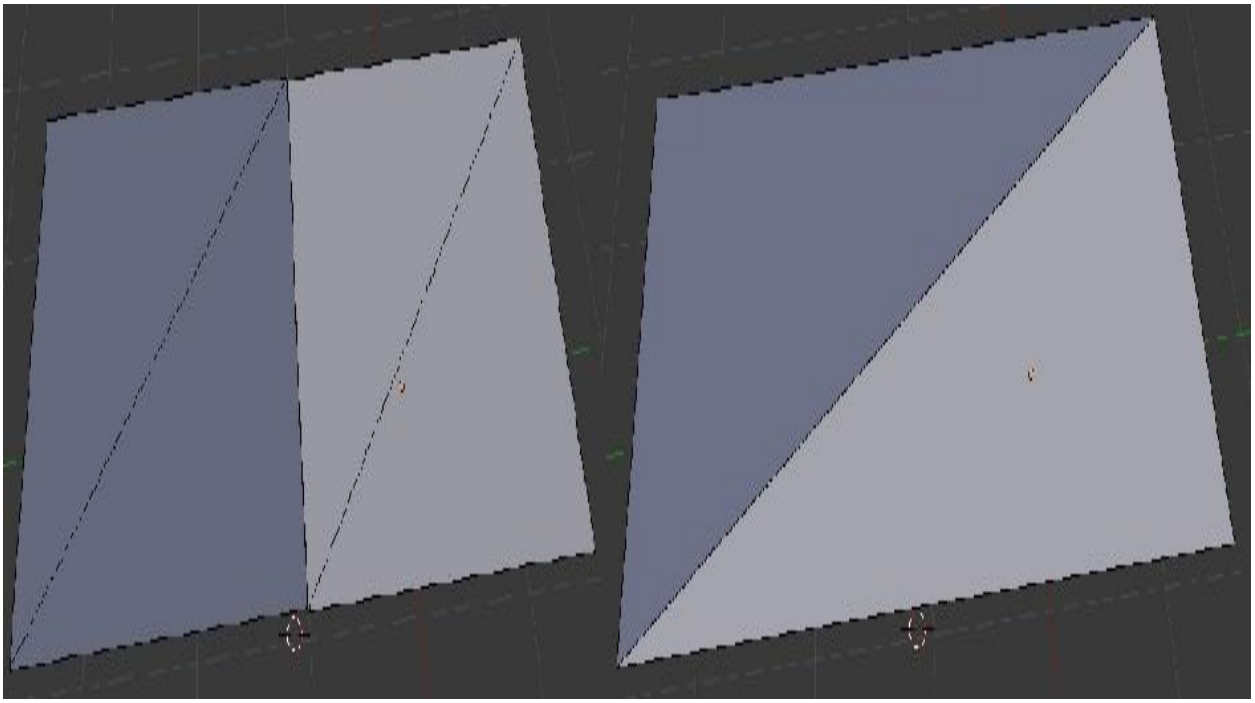

Рисунок 2.14 - Один объект с разным количеством фасетов

Рассмотрим ошибку в результате обхода на примере рисунке 2.15. Начальный фасет у нас 1. Фасеты 2 и 4 являются смежными. Значения в

трехмерной таблице для фасета 4 увеличивается на 1. Для фасета 2 остается без изменения, так как он и фасет 1 лежат в одной плоскости. На следующем шаге мы переходим на фасет 2. Фасет 1 и 3 являются смежными. Фасет 1 уже участвовал в анализе, поэтому значение в таблице не изменяется. Фасет 3 лежит в той же плоскости, что и фасет 2. На 3 шаге главный фасетов становиться фасет 3. Фасет 2 не рассматривается, так как уже участвовал в анализе. Фасет 5 является смежным и лежит в другой плоскости. Увеличиваем значение в таблице для фасета 4 на единицу. Но в тоже время фасет 4 и 5 лежит в одной плоскости. В результате получаем ошибку в расчетах таблицы.

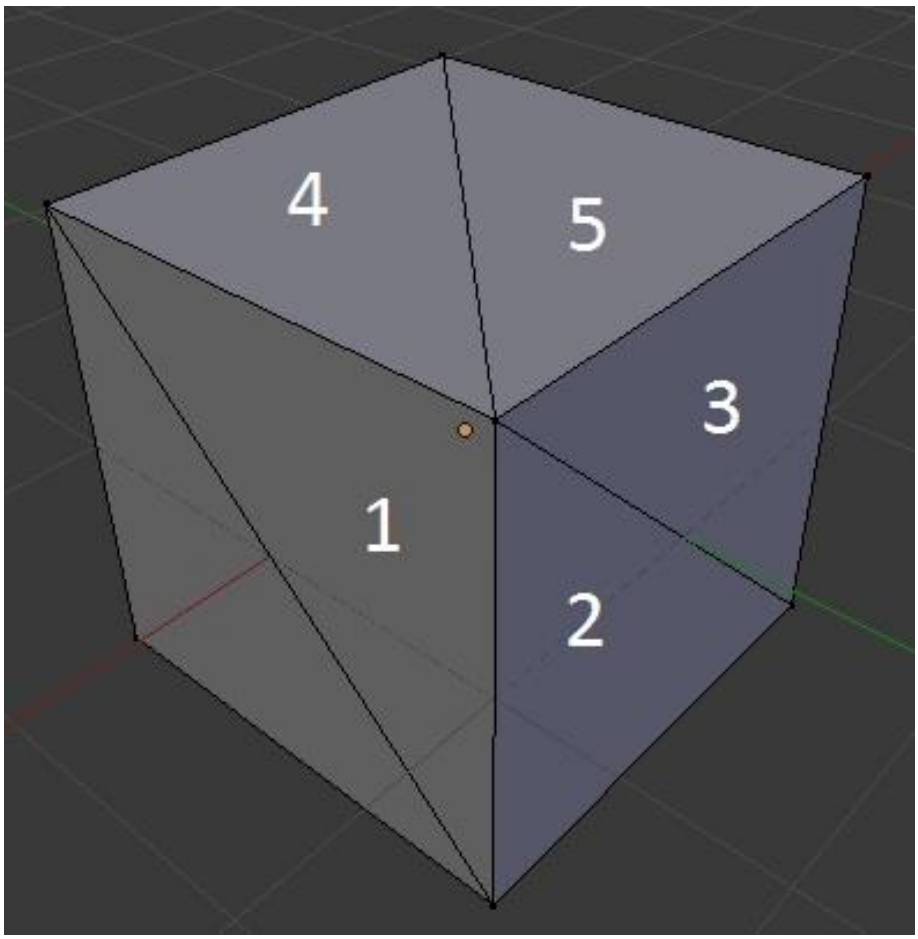

Рисунок 2.15 - Проблема обхода

Чтобы избежать ошибки расчетов, при обходе фасетов, сначала будем обходить все смежные фасеты, лежащие в одной плоскости, а потом уже остальные. Значение в таблице будем увеличивать, когда начнем рассматривать новую группу фасетов, лежащих в одной плоскости. То есть, сначала рассмотрим фасет 1. Увеличим значение в таблице для фасета 1. Рассмотрим

все смежные с ним фасеты. После того, как вся группа смежных фасетов будет рассмотрена, переходим к следующему фасеты, не участвующему в анализе. То есть к фасету 2.

На рисунке 2.16 показан, как будет происходить обход куда, который не приведёт к ошибке. Сначала мы проанализируем красную грань, потом зелёную и синюю. А далее остальные грани. Несмотря на всё это у нас остается проблема сравнения двух таблиц.

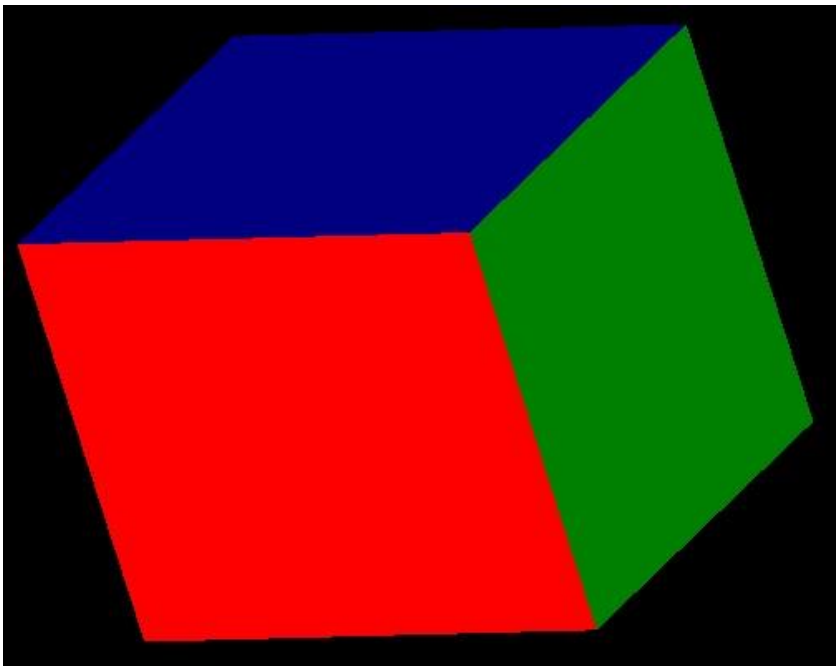

Рисунок 2.16 - Пример обхода фасетов

На рисунке 2.17 показаны две одинаковые пирамиды. Если мы их будем анализировать, то получим перевёрнутые таблицы, несмотря на то, что это одинаковые объекты. Поэтому для того, чтобы сравнить две таблицы, мы будем прообразовывать одну, до достижения максимальной схожести с другой.

Для достижения максимальной схожести двух таблиц, мы сначала выберем одно из значений в трехмерной таблице. Лучше всего, чтобы это было значение наиболее отличное от других. После чего во второй таблице найдет такое же число. Если оно отсутствует, то таблицы не схожи. Если число будет найдено, продолжим вычисления. При помощи поворотов таблицу по осям x, y, z, мы выставляем два числа в одинаковое положение в двух таблицах. Далее смотрим смежные числа с выбранным в начале числом. Если все числа стоят на

своих местах, то проверяем таблицу на схожесть всех значений. Если нет, то мы вращаем числа вокруг начального, пока числа не совпадут или не закончатся возможности положений чисел. Если такое положение не будет найдено, то таблицы не одинаковые. Если в таблице несколько наиболее отличных значений, то мы проверяем их по очереди, пока не найдем схожесть таблиц или не убедимся в обратном решении.

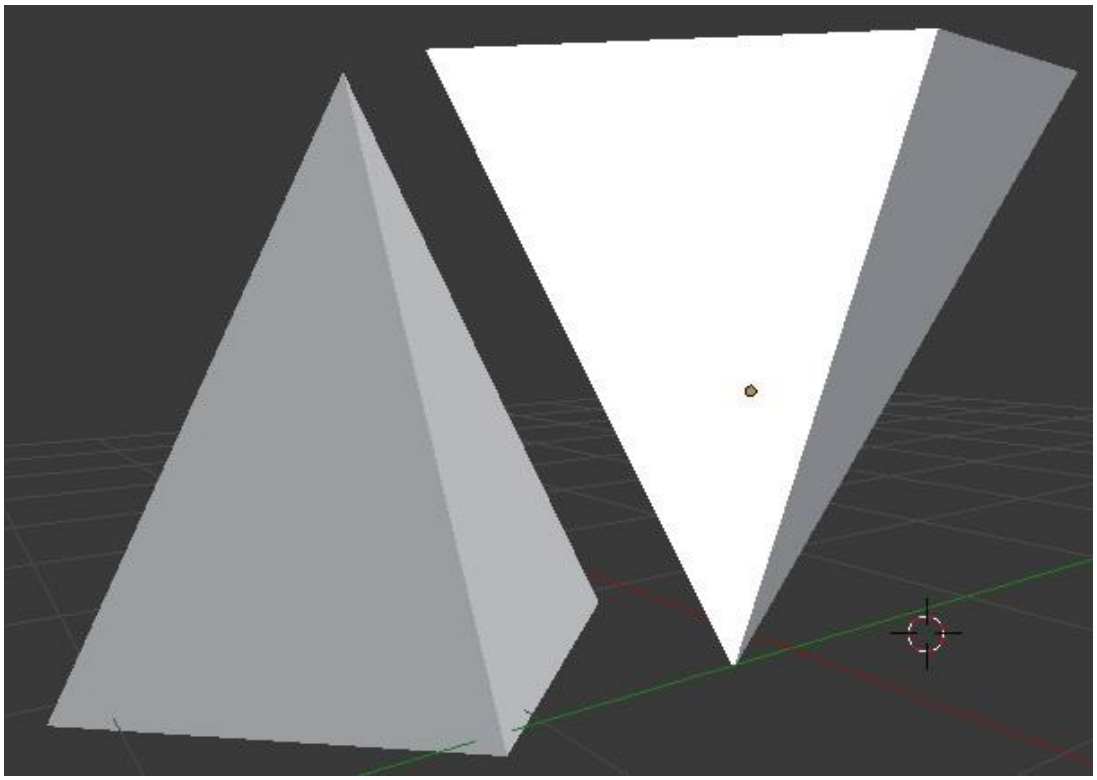

Рисунок 2.17 - Трехмерная модель двух одинаковых пирамид

Таким образом, мы можем сравнить две таблицы. Если задать параметр схожести объектов, то при анализе таблиц и с учетом этого параметра, можно будет найти схожие объекты.

Данный алгоритм сравнения таблиц хоть и позволяет нам анализировать форму объекта, но не дает нам полной картины. В результате работы алгоритма, в таблице храниться упрощенная модель трехмерного объекта. Мы не можем по трехмерной таблице воссоздать трехмерную модель. Поскольку нам известно как фасеты, записанные в таблице, должны располагаться в трехмерном пространстве.

Одним из вариантов решения данной проблемы, является сохранение ссылки, на смежный фасет, при программировании. То есть когда будет происходить обход по области, надо запоминать, какая область лежит рядом. Зная это, мы сможем воссоздать примерную форму трехмерной модели, анализ которой проводился.

Глава 3 Реализация программы для демонстрации работы алгоритмов

3.1 Разработка архитектуры ПО

Для демонстрации работоспособности алгоритмов, разработаем приложение. При реализации проекта будем использовать клиент-серверную архитектуру.

Выбор языка программирования, на котором будет реализован клиент и сервер, не имеет значения большого. Главное, чтобы язык программирования поддерживал работу с трёхмерной графикой. Сейчас работе с трехмерной графикой ведётся повсеместно. Процесс создания приложений в настоящее время упрощён. Поэтому мы воспользуемся фреморком QT Creator для ускорения создания приложения. Данный фрейморк написан на языке C++. В нем уже присутствуют всё необходимые функции для работы с трехмерной графиков. Так же в нем есть множество библиотек, позволяющих отправлять запросы на сервер и подключаться к базам данных.

В качестве СУБД будем использовать MySQL. MySQL – это бесплатная СУБД от Oracle. Она проста в использовании и легко настраивается. MySQL является одной и самых популярный СУБД в интернете. Она отлично подходит для маленьких и больших проектов. В будущем, при создании крупной системы, все данных, которые будут содержаться в MySQL, можно будет легко перенести в другую СУБД.

3.2 Разработка базы данных

Хранение STL файлов ведётся уже давно. Но в нашей системе нам необходимо хранить данные анализа, а так же разбирать это данные, по запросу пользователя.

Разработка баз данных так же является процессом, который реализуется повсеместно. Для разработки БД сейчас существует множество инструментов, которые позволяют создать её, даже неопытным пользователям. Сейчас множество интернет сервисов предоставляют такую возможность. Достаточно просто схематически нарисовать базу данных, после чего сервис уже выдаст

готовый файл sql, содержащий все данных, необходимые для построения БД. На рисунке 3.1 представлена готовая база данных, созданная для разрабатываемой системы.

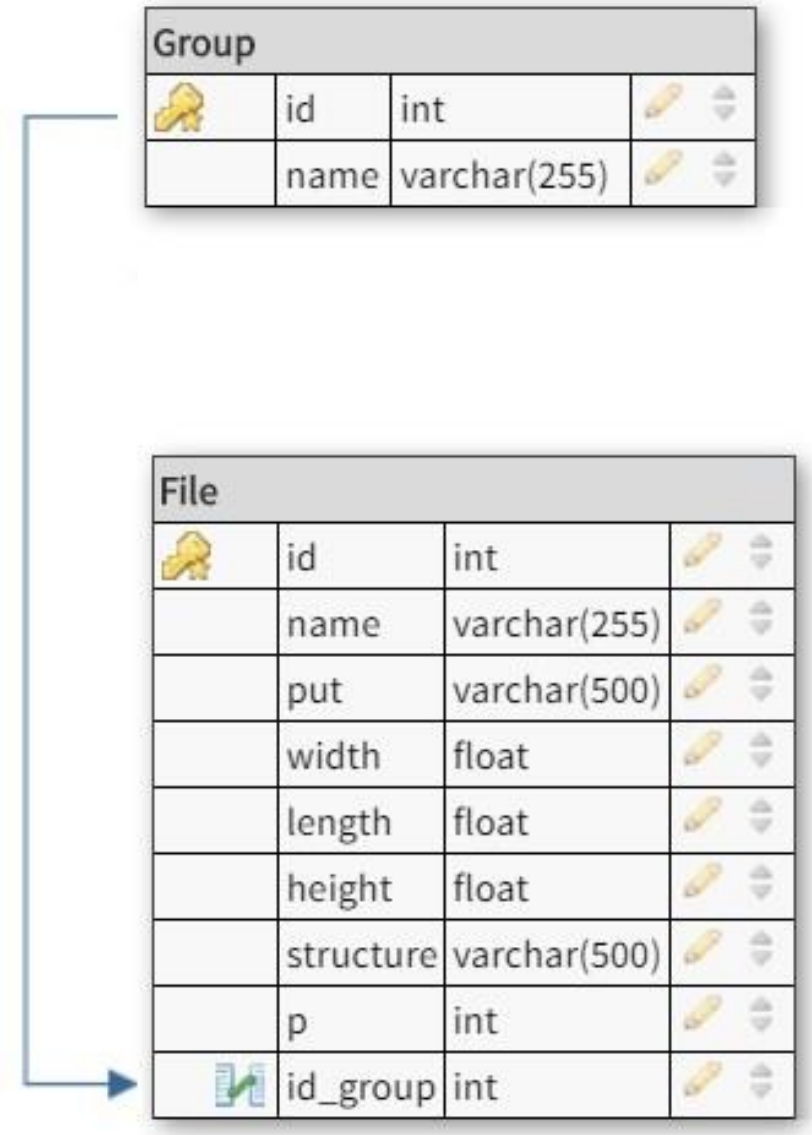

Рисунок 3.1 - Структура базы данных

Таблица Group содержит все данные о группах классификации STL файлов. Поле id – уникальный номер группы. Поле name – содержит название категории и ограничено 255 символами.

Таблица File будет содержать путь до STL файлов и данные о файле. Поле name – название объекта, который описан при помощи STL файла. Поскольку названия объектов могут повторяться, в таблице введено поле id, которое однозначно определяет каждый файл. Поле put – содержит путь до

файла. Файлы STL на компьютере могут располагаться в разных папках. Поэтому необходимо обязательно запоминать путь до файла, чтобы быстро находить его в системе. Поле width – ширина объекта. Поле height – высота объекта. Поле length – длина объекта. Поле p – показывает род объекта. Поле structure – содержит в себе трёхмерную таблицу объекта.

3.3 Разработка программы

Как было сказано ранее, при разработке ПО мы будем использоваться фреймворк QT Creator. Его основная особенность – это способ общения между объектами. Общение происходит по средствам сигналов и слотов. Ожидание сигнала на слот реализовано в виде бесконечного цикла. На рисунке 3.2 показано алгоритм работы глобального цикла QT Creator. В данном алгоритме бесконечный цикл подразумевает под собой ожидание сигнала на окончание работы приложения.

Рисунок 3.2 - Глобальный цикл QT Creator

Опишем основной алгоритм работы программы внутри глобального цикла. Алгоритм представлен на рисунке 3.3.

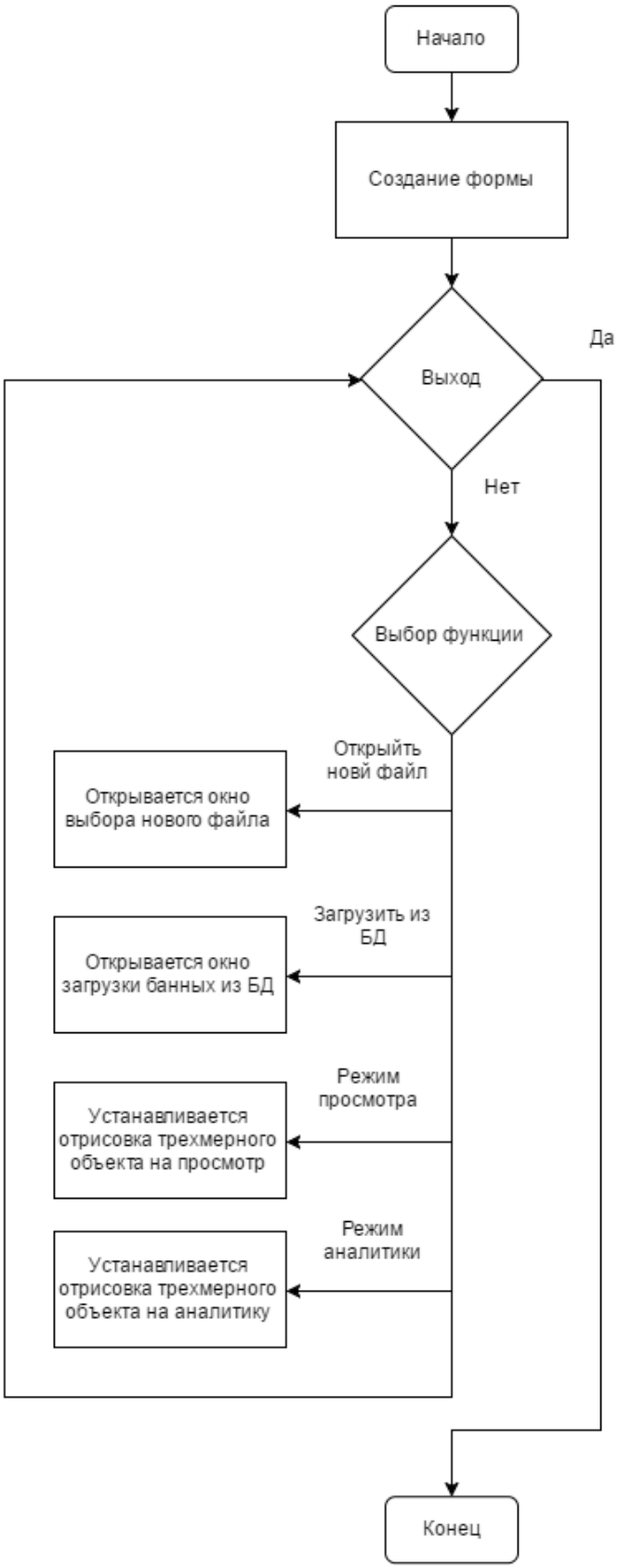

Рисунок 3.3 - Основой алгоритм работы программы

На рисунке 3.4 представлен алгоритм анализа формы трехмерных объектов.

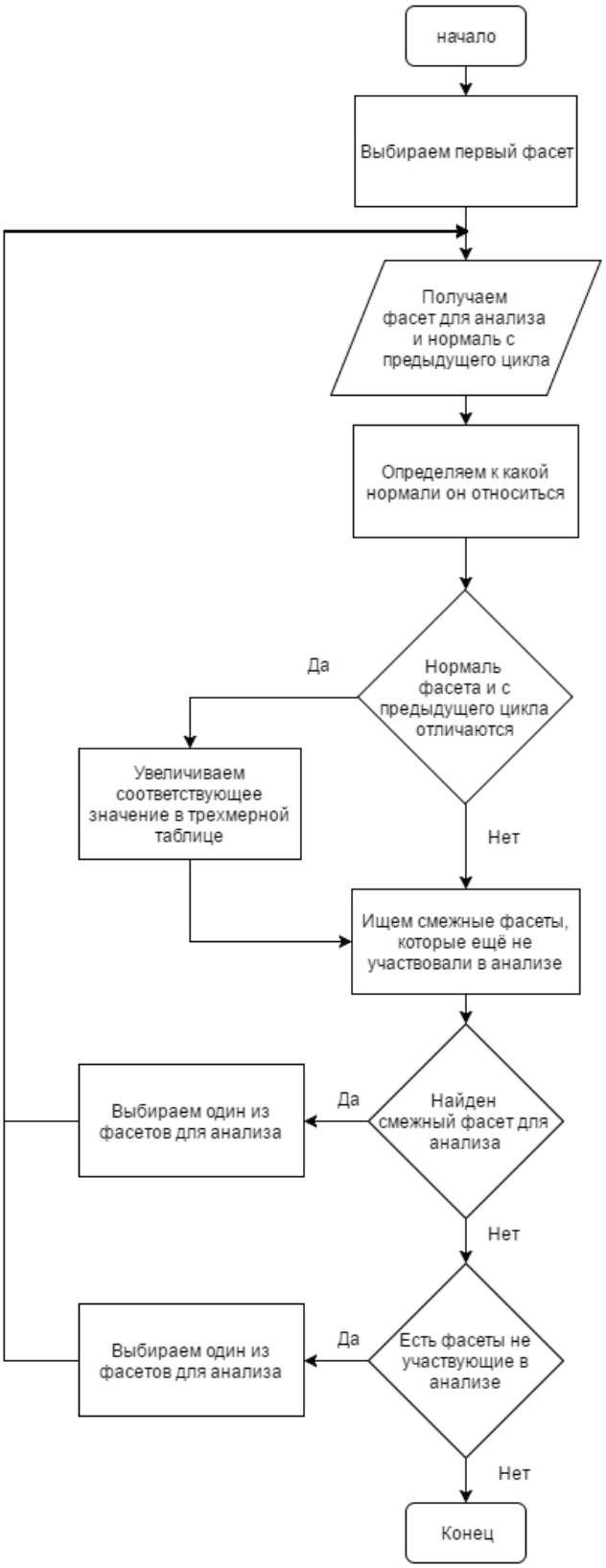

Рисунок 3.4 - Алгоритм анализа формы трехмерной модели

Каждый объект в QT Creator имеет свой набор сигналов и слотов. Так же можно добавлять свои сигналы и слоты. Система добавления слотов, сигналов дает большую гибкость при разработке системы.

На рисунке 3.5 показан начальный экран программы. На нем отображена трехмерная модель загруженного файла. В правой части экрана находится поле, для ввода данных программистом, а так же трехмерное отображение трехмерной таблицы.

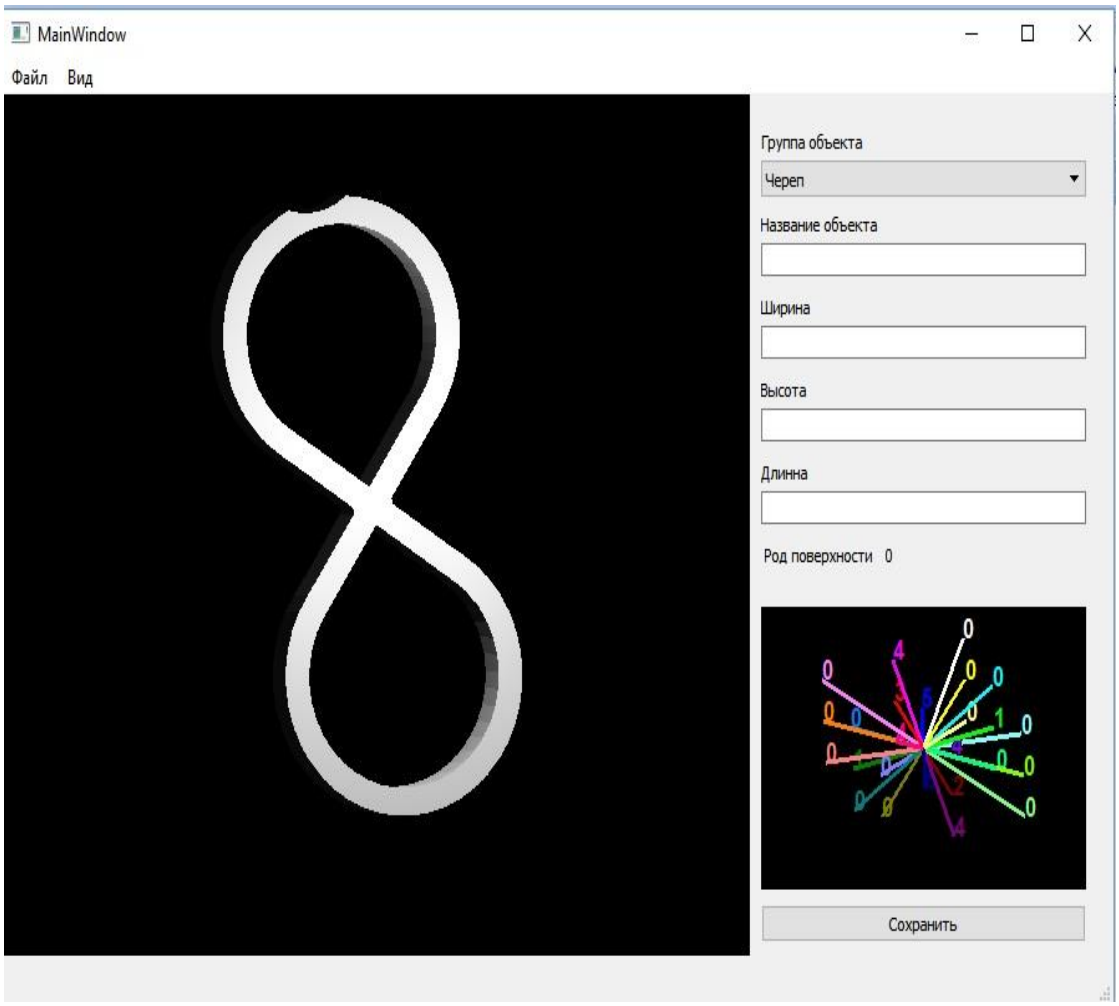

Рисунок 3.5 - Начальное окно программы

В программе предусмотрена возможность вращать трехмерные модели и таблицу.

В верхнем левом углу расположено меню выбора файла для открытия, показанное на рисунке 3.6.

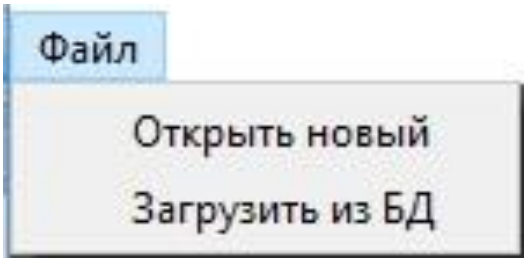

Рисунок 3.6 - Верхнее меню выбора файла

В данном меню предоставлена возможность открыть новый файл или загрузить объект из базы данных. Диалоговое окно выбора нового STL файла представлено на рисунке 3.7.

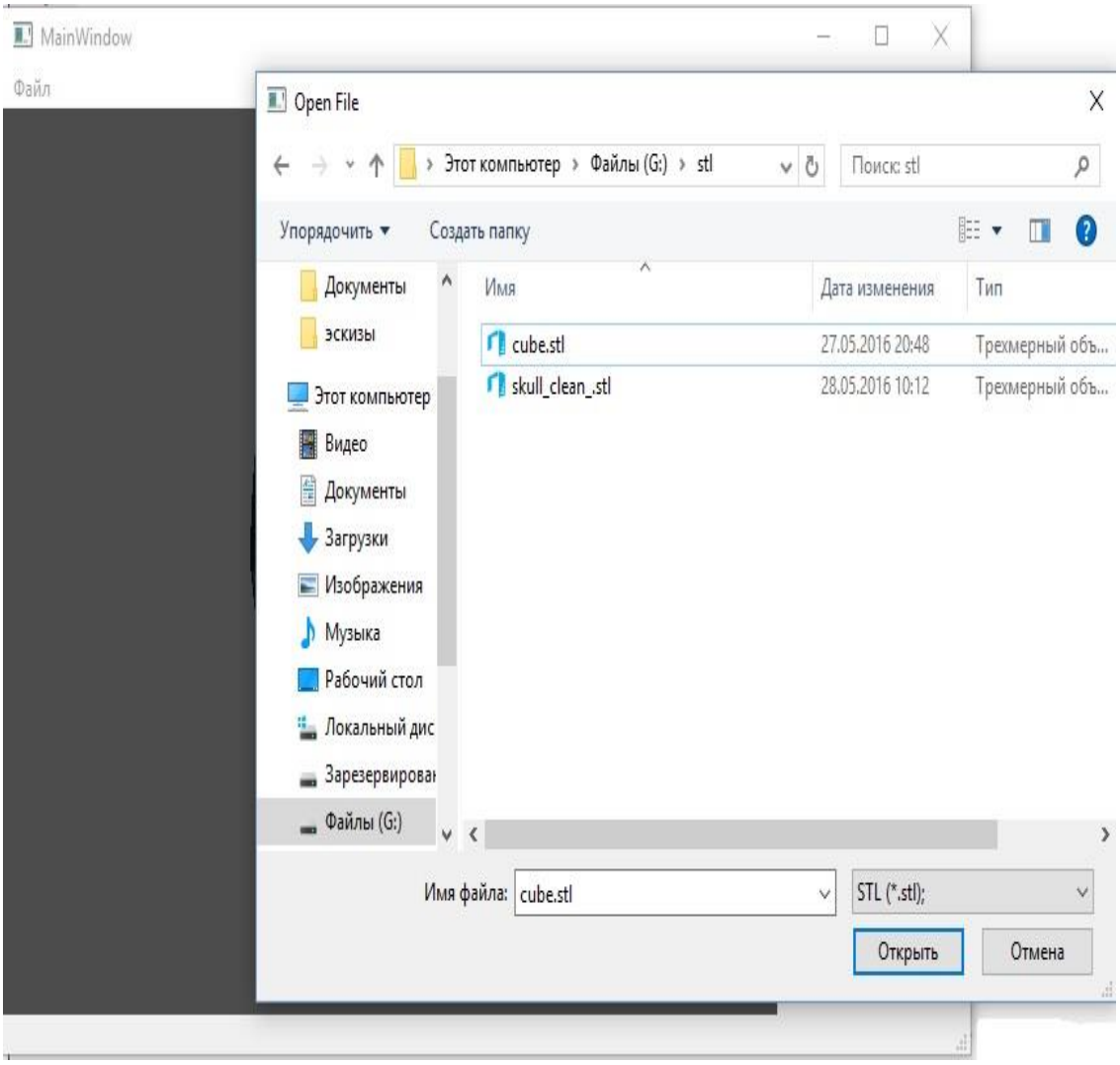

Рисунок 3.7 - Диалоговое окно открытие нового файла

В нем присутствует фильтрация файлов по формату, что позволяет ускорить поиск нужного файла среди других файлов.

При выборе меню «Загрузить из БД» открывается диалоговое окно, показанное на рисунке 3.8. В данном меню можно отфильтровать файлы по

группе. Так же в данном меню можно добавить новую группу, в которую можно будет добавить новые файлы.

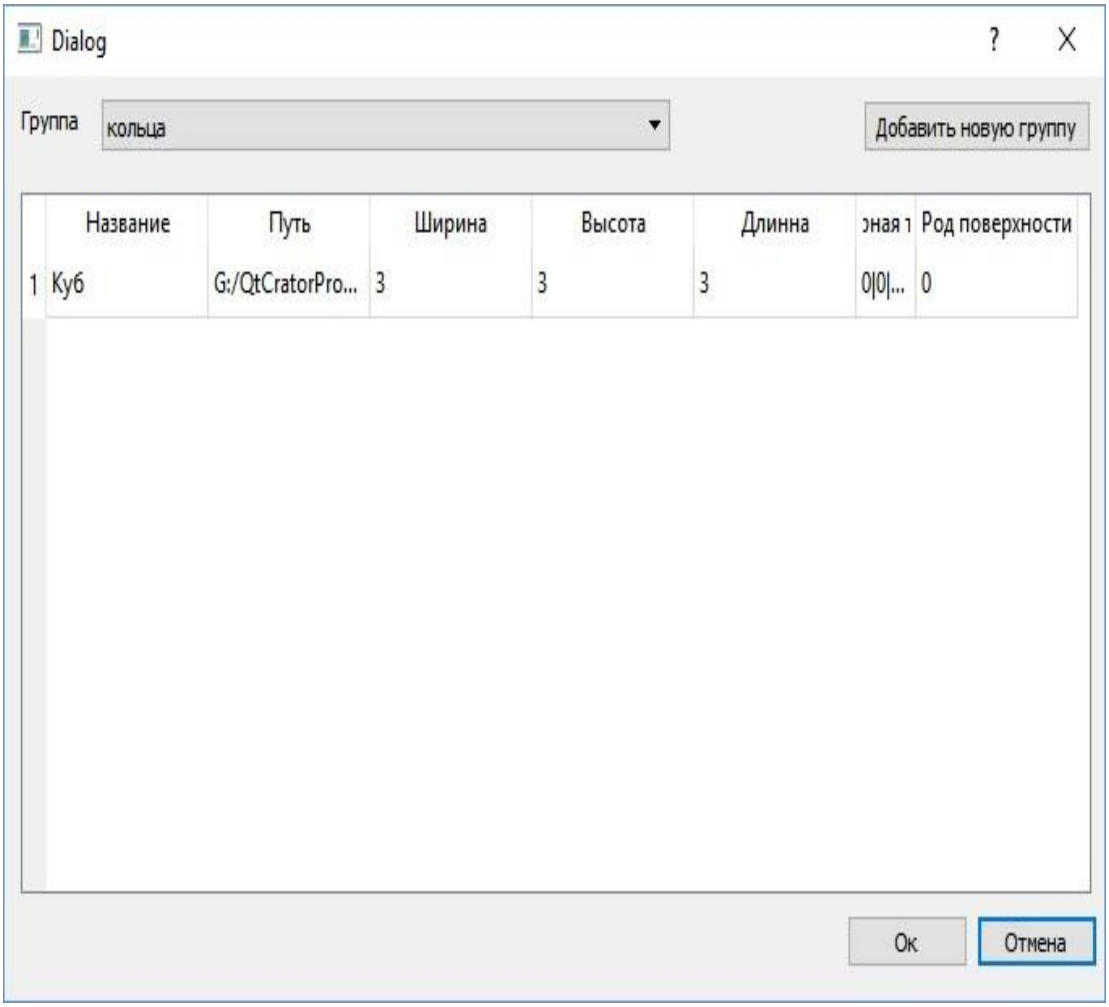

Рисунок 3.8 - Диалоговое окно загрузки файла из базы данных

Программа имеет два режима просмотра, «Аналитика» и «Просмотр». Меню выбора способа просмотра показано на рисунке 3.9.

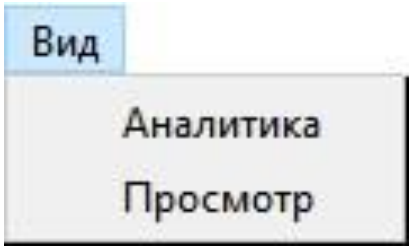

Рисунок 3.9 – Меню выбора вида отображения трехмерного объекта

В режиме аналитики нам показывается, как анализируется файл. Пример режима аналитики показан на рисунке 3.10.

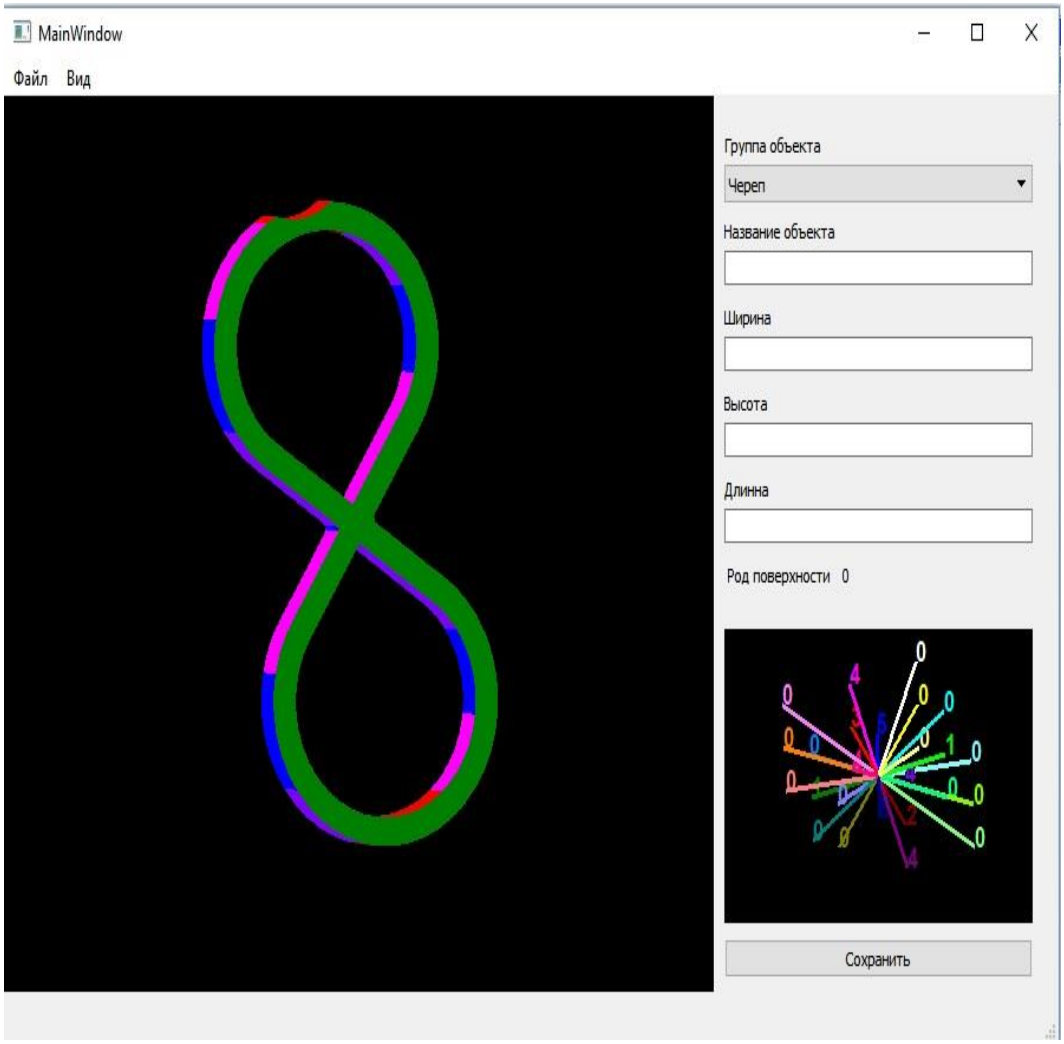

Рисунок 3.10 – Программа в режиме аналитики

Разработанный программный продукт отвечаем всем требования, поставленный перед нами.

Заключение

В результате выполненной бакалаврской работы был рассмотрен способ создания твердотельных объектов. Разобрана структура STK файлов и проведен их анализ.

Разработанная база данных включаем наиболее полную и необходимую информацию об STL файле. Она позволит находить файлы по критериям твердотельной модели, которая в результате может быть получена.

Топологический анализ позволил получить данных из STL файла, которые не изменяются в ходе деформации. При топологическом анализе мы получаем род поверхности, который показывает нам количество «дырок» в структуре файла.

При помощи эвристического анализа мы получаем информацию о структуре формы объекта. В будущем, при анализе этих данных можно будет находить схожие объекты по форме. Используя повороты трехмерной таблицы, мы можем находить одинаковые объекты, даже если изначально они были повернуты.

Для демонстрации поделанной работы была разработана программа. В ней предоставляется возможность предварительного просмотра файла, открытие нового файла, и сохранение его базе данных. В программе присутствует возможность просмотра составленной трехмерной таблицы, полученной в результате анализа. Так же в программе присутствует возможность включить режим анализа, что демонстрации того, как проходил анализ формы по эвристическому алгоритму.

Несмотря на то, что в будущем, из-за большого количества ошибок в STL файла, формат файла может измениться, алгоритм будет работать. Так же его можно модифицировать, для анализа не только STL файлов, а для файлов любого формата, структура которых основана на хранении полигонной сетки.

Организация анализа и хранения STL файлов позволит в будущем избавиться от одинаковых файлов. Так же в будущем, по данным, можно будет организовать поиск нужных файлов. Это увеличит скорость воспроизводства

новых или поиск старых файлов, если необходимо снова воссоздать старый объект. Анализ формы трехмерных объектов и поиск по ним позволит лучше следить за правами на трехмерные модели. Из-за отсутствия систему поиска, эти права часто нарушаются.

Список используемой литературы

Учебники и учебные пособия

1. Аббасов И.Б. Компьютерное моделирование в промышленности – М.: ДМК Пресс – 2013 – 92 с.

2. Босс В. Лекции по математике. Т. 13: Топология: Учебное пособие. — М.: Книжный дом «ЛИБРОКОМ», 2009. — 216 с.

3. Виро О. Я., Иванов О. А., Нецветаев Н. Ю., Харламов В. М. Элементарная топология. - М.: МЦНМО, 2007. - 446 с.

4. Иванов А. О., Ильютко Д. П., Носовкий Г. В., Тужилин А. А., Фоменко А. Т. Компьютерная геометрия: практикум. Учебное пособие. – М.: БИНОМ, Интернет-Университет Информационных Технологий, 2010. - 392 с.

5. Иванов А. О., Тужилин А. А. Лекции по классической дифференциальной геометрии – М.: Логос, 2009 – 180 с.

6. Иванов А. О., Тужилин А. А. Лекции по классической дифференциальной геометрии. Учебное пособие. – М.: Логос, 2009. - 224 с.

7. Матвеев С. В. Алгоритмическая топология и классификация трехмерный многообразий – М.: МЦНМО, 2007 – 456 с.

8. Мищенко А. С., Фоменко А. Т.. Курс дифференциальной геометрии и топологии. Учебник, переаботанное и дополненное издание. – М.: Лань, 2010. - 503 с.

9. Ошемков А. А., Попеленский Ф. Ю., Тужилин А. А., Фоменко А. Т., Шафаревич А. И.. Курс наглядной геометрии и топологии. — В серии: Классический учебник МГУ. – М.: URSS, ООО ЛЕНАНД. 2015 - 360 с.

10. Прахов А. А. Blender. 3D моделирование и анимация – Спб.: БХВ, 2009 – 227 с.

11. Примаков Д. А., Хамидуллин Р. Я. Геометрия и топология – Спб.: Маркет Дс, 2011 - 272 с.

12. Розендорн Э. Р. Теория поверхностей. — 2-е изд., перераб. и доп. – М.: ФИЗМАТЛИТ, 2006. - 304 с.

13. Тужилин А. А., Фоменко А. Т.. Элементы геометрии и топологии

минимальных поверхностей. — В серии: Классический учебник МГУ. Издание второе, дополненное. – М.: URSS, ООО ЛЕНАНД. 2014 - 256 с.

14. Фоменко А. Т., Фукс Д. Б.. Курс гомотопической топологии. — В серии: Классический учебник МГУ. Издание 2-е. – М.: URSS, ООО ЛЕНАНД. 2014 - 512 с.

15. Чернавский А. В., Мищенко Е. Ф. Геометрическая топология, дискретная геометрия и теория множеств – М.: МАИК "Наука / Интерпериодика", 2006 – 287 с.

Электронные ресурсы

16. Токарев Б. Е. 3D печать: рынок и перспективы 2014 // Высшая школа маркетинга и развития бизнеса [Электронный ресурс]: сайт: http://www.bioprinting.ru/upload/iblock/333/333c1ba4f5c6a8b477a3d7aca49ce4a7.pdf

Литература на иностранном языке

17. Mr. Kalpesh Sarode ; Prof Sudhir Chaurey ; Prof Anwesh Virkunwar, REVIEW PAPER: ANALYSIS OF DATA TRANSFORMATION IN RAPID PROTOTYPING TECHNOLOGY, Scientific Journal Impact Factor, April, 2015, ISSN: 2277-9655

18. Prof. Edward Aboufadel Department of Mathematics Grand Valley State University, 3D Printing A Pendant with A Logo, Scientific Journal Impact Factor, July 2015, ISSN: 2277-9655

19. K. Murugesan, Ponsekar Abraham Anandapandian, Sumeet Kumar Sharma , M. Vasantha Kumar, Comparative Evaluation of Dimension and Surface Detail Accuracy of Models Produced by Three Different Rapid Prototype Techniques, 21 September 2011

20. Wentao Zha, Geometric Approaches to Input File (STL) Modification for Part Quality Improvement in Additive Manufacturing, December 2015

21. Tsung-Yu Lee, Ren-Guey Lee, Sheng-Chung Tien, Robert Lin, Wei-Hua Su, An Accelerating 3D Image Reconstruction System Based on the Level-of-Detail Algorithm, December 2012

Приложение А

Листинги кода реализации классов

Класс main

```
#include "mainwindow.h"
#include <QApplication>
#include <mn.h>
int main(int argc, char *argv[])
{
    QApplication :: setAttribute(Qt :: AA_UseDesktopOpenGL);
    QApplication a(argc, argv);

    mn w;
    w.show();

    return a.exec();
}
```

Класс mn

```
#ifndef MN_H
#define MN_H

#include <QMainWindow>
#include <stl.h>

namespace Ui {
class mn;
}

class mn : public QMainWindow
{
    Q_OBJECT

public:
    explicit mn( QWidget *parent = 0);
    ~mn();
    QString GetRandomString();

    void loadFileDB ( QString put, STL* stl);

private slots:
    void on_action_triggered();

    void on_action_2_triggered();

    void on_pushButton_clicked();

    void on_action_3_triggered();

    void on_action_4_triggered();

private:
    Ui::mn *ui;
};

#endif // MN_H
#include "mn.h"
#include "ui_mn.h"
#include <QFileDialog>

#include <QSqlDatabase>
#include <QDebug>
#include <mn.h>
#include <loaddatadb.h>
#include <QSqlQuery>

mn::mn(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::mn)
{
    ui->setupUi(this);

    QDir* dir = new QDir ( QCoreApplication :: applicationDirPath() + "/stl");
if(!dir->exists())
        dir->mkdir( QCoreApplication :: applicationDirPath() + "/stl"  );
```

```
    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    db. setHostName("127.0.0.1");
    db. setPort(3306);
    db. setDatabaseName("diplom");
    db. setUserName("root");
    db. setPassword("qwerty");

    if(db.open()){
        qDebug() << "Not error";
    } else {
        qDebug() << "Error";
    }

    QSqlQuery query;
    query.exec("SELECT `id`, `name` FROM `group`");
    while (query.next()) {
        int id =  query.value(0).toInt();
        QString name = query.value(1).toString();
        ui->comboBox->addItem( name , QVariant( QString::number(id) ));
    }

    ui->widget_2->restartTable( ui->widget->stl->table);
    ui->p->setText( QString::number(ui->widget->stl->p));
}

mn::~mn()
{
    delete ui;
}

void mn::on_action_triggered()
{
    QFileDialog dialog;
    QString file = dialog. getOpenFileName(
                this,
                tr("Open File"),
                "",
                tr("STL (*.stl);" )
                );

    if (file != ""){
        ui->widget->ReadBinary( file );
        ui->widget->stl->put = file;
        ui->widget->stl->analisis();
        ui->widget_2->restartTable( ui->widget->stl->table);
    }
}

void mn::on_action_2_triggered()
{
    LoadDataDB* group = new LoadDataDB(this);
    group->show();
}

QString mn::GetRandomString()
{
   const QString
   QString randomString;
   for(int i=0; i<randomStringLength; ++i)
   {
       int index = qrand() % possibleCharacters.length();
       QChar nextChar = possibleCharacters.at(index);
       randomString.append(nextChar);
   }
   return randomString;
}

void mn::on_pushButton_clicked()
{
    QFile *file = new QFile( ui->widget->stl->put );
    QFileInfo* fInfo = new QFileInfo( ui->widget->stl->put );
    if (!file->exists()) {
         return;
   } else {
        QDateTime dateTime = dateTime.currentDateTime();
        QString dateTimeString = dateTime.toString("yyyy-MM-dd_hh_mm_ss")
QString file_put = QCoreApplication::applicationDirPath() + "/stl/" + GetRandomString() +
dateTimeString + "." +fInfo->completeSuffix();
```

```
        bool cop = file->copy( file_put );
        qDebug() << cop;
        ui->widget->stl->put = file_put;
    }

    ui->widget->stl->idgroup = ui->comboBox->itemData( ui->comboBox->currentIndex()).toInt();
    ui->widget->stl->name = ui->name->text();
    ui->widget->stl->height = ui->height->text().toFloat();
    ui->widget->stl->width = ui->width->text().toFloat();
    ui->widget->stl->length = ui->lenght->text().toFloat();
    ui->widget->stl->save();
}

void mn::loadFileDB(QString put, STL* stl){
    ui->widget->ReadBinary( put );

    ui->name->setText(stl->name);
    ui->width->setText(QString::number(stl->width));
    ui->height->setText(QString::number(stl->height));
    ui->lenght->setText(QString::number(stl->length));
    ui->p->setText(QString::number(stl->p));

    ui->widget->stl->name = stl->name;
    ui->widget->stl->width = stl->width;
    ui->widget->stl->height = stl->height;
    ui->widget->stl->length = stl->length;
    ui->widget->stl->p = stl->p;
    ui->widget_2->restartTable( ui->widget->stl->table);
}

void mn::on_action_3_triggered()
{
    ui->widget->analiticView = true;
    ui->widget->returView();
}

void mn::on_action_4_triggered()
{
    ui->widget->analiticView = false;
    ui->widget->returView();
}
```

Класс loaddatadb

```
#ifndef LOADDATADB_H
#define LOADDATADB_H

#include <QDialog>

namespace Ui {
class LoadDataDB;
}

class LoadDataDB : public QDialog
{
    Q_OBJECT

public:
    explicit LoadDataDB(QWidget *parent = 0);
    ~LoadDataDB();

private slots:
    void on_pushButton_3_clicked();

    void on_comboBox_currentIndexChanged(int index);

    void on_pushButton_2_clicked();

    void on_pushButton_clicked();

private:
    Ui::LoadDataDB *ui;
};
#endif // LOADDATADB_H

#include "loaddatadb.h"
#include "ui_loaddatadb.h
#include <QSqlQuery>
#include <QDebug>
```

```
#include <QInputDialog>

#include <mn.h>
#include <stl.h>
#include <QTableWidgetItem>

LoadDataDB::LoadDataDB(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::LoadDataDB)
{
    ui->setupUi(this);

    QSqlQuery query;
    query.exec("SELECT `id`, `name` FROM `group`");
    while (query.next()) {
        int id =  query.value(0).toInt();
        QString name = query.value(1).toString();
        ui->comboBox->addItem( name , QVariant( QString::number(id) ));
    }

//    QString::number(ui->comboBox->itemData( ui->comboBox->currentIndex()))

    QString idGroup = QString::number( ui->comboBox->itemData( ui->comboBox->currentIndex()).toInt()
);

    ui->tableWidget->setColumnCount(7);
    ui->tableWidget->setHorizontalHeaderItem(0, new QTableWidgetItem(tr("Название")));
    ui->tableWidget->setHorizontalHeaderItem(1, new QTableWidgetItem(tr("Путь")));
    ui->tableWidget->setHorizontalHeaderItem(2, new QTableWidgetItem(tr("Ширина")));
    ui->tableWidget->setHorizontalHeaderItem(3, new QTableWidgetItem(tr("Высота")));
    ui->tableWidget->setHorizontalHeaderItem(4, new QTableWidgetItem(tr("Длинна")));
    ui->tableWidget->setHorizontalHeaderItem(5, new QTableWidgetItem(tr("Трехмерная таблица")));
    ui->tableWidget->setHorizontalHeaderItem(6, new QTableWidgetItem(tr("Род поверхности")));

    ui->tableWidget->setSelectionBehavior(QAbstractItemView::SelectRows);


    query.exec("SELECT name, put, width, height, length, structure, p FROM file where idgroup =
"+idGroup+" ");

    QTableWidgetItem* item;

    for(int i=ui->tableWidget->model()->rowCount()-1; i >= 0; --i)
         ui->tableWidget->removeRow(i);

    while (query.next()) {
        ui->tableWidget->insertRow( ui->tableWidget->rowCount() );
        item = new QTableWidgetItem( query.value(0).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 0, item);

        item = new QTableWidgetItem( query.value(1).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 1, item);

        item = new QTableWidgetItem( query.value(2).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 2, item);

        item = new QTableWidgetItem( query.value(3).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 3, item);

        item = new QTableWidgetItem( query.value(4).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 4, item);

        item = new QTableWidgetItem( query.value(5).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 5, item);

        item = new QTableWidgetItem( query.value(6).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 6, item);
    }
}

LoadDataDB::~LoadDataDB()
{
    delete ui;
}void LoadDataDB::on_pushButton_3_clicked()
{
    bool bOk;
    QString str = QInputDialog::getText( 0,
                                         "Input",
                                            "Название группы:",
```

```
                                        QLineEdit::Normal,
                                        "",
                                        &bOk
                                       );
    if (bOk) {
        QSqlQuery query;
        query.exec("INSERT INTO `group` (`name`) VALUES ('"+ str +"');");
        int key = query.lastInsertId().toInt();
        ui->comboBox->addItem( str , QVariant( QString::number(key) ));
    }
}

void LoadDataDB::on_comboBox_currentIndexChanged(int index)
{
    QString idGroup = QString::number( ui->comboBox->itemData( ui->comboBox->currentIndex()).toInt()
);

    QSqlQuery query;
    query.exec("SELECT name, put, width, height, length, structure, p FROM file where idgroup =
"+idGroup+" ");

    for(int i=ui->tableWidget->model()->rowCount()-1; i >= 0; --i)
         ui->tableWidget->removeRow(i);

    QTableWidgetItem* item;

    while (query.next()) {
        ui->tableWidget->insertRow( ui->tableWidget->rowCount() );
        item = new QTableWidgetItem( query.value(0).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 0, item);

        item = new QTableWidgetItem( query.value(1).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 1, item);

        item = new QTableWidgetItem( query.value(2).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 2, item);

        item = new QTableWidgetItem( query.value(3).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 3, item);

        item = new QTableWidgetItem( query.value(4).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 4, item);

        item = new QTableWidgetItem( query.value(5).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 5, item);

        item = new QTableWidgetItem( query.value(6).toString() );
        ui->tableWidget->setItem( ui->tableWidget->rowCount() -1 , 6, item);
    }
}

void LoadDataDB::on_pushButton_2_clicked()
{
    mn *w = qobject_cast<mn*>(parent());

    int row = ui->tableWidget->currentIndex().row();

    STL* stl = new STL();

    stl->name = ui->tableWidget->item( row, 0)->text();
    stl->width = ui->tableWidget->item( row, 2)->text().toDouble();
    stl->height = ui->tableWidget->item( row, 3)->text().toDouble();
    stl->length = ui->tableWidget->item( row, 4)->text().toDouble();
    stl->loadTable( ui->tableWidget->item( row, 5)->text() );
    stl->p = ui->tableWidget->item( row, 6)->text().toInt();

    QString put = ui->tableWidget->item( row, 1)->text();
    w->loadFileDB( put, stl);

}

void LoadDataDB::on_pushButton_clicked()
{
    close();
}
```

Класс vertex

```
#ifndef VERTEX_H
```

```
#define VERTEX_H


class vertex
{
public:
    double x = 0.0, y = 0.0, z = 0.0;
    vertex();
    vertex(double x, double y, double z);
    double Distance(vertex* fas);
    friend bool operator==(const vertex& left, const vertex& right);
    float ugol(vertex *sm);

    vertex& operator=(const vertex& right) {
            //проверка на самоприсваивание
            if (this == &right) {
                return *this;
            }
            this->x = right.x;
            this->y = right.y;
            this->z = right.z;
            return *this;
        }
};

#endif // VERTEX_H


#include "vertex.h"
#include <math.h>

#include <QDebug>

vertex::vertex()
{
}

vertex::vertex(double x, double y, double z)
{
    this->x = x;
    this->y = y;
    this->z = z;
}

bool operator==(const vertex& left, const vertex& right) {
    if(left.x == right.x &&
       left.y == right.y &&
       left.z == right.z){
       return true;
    }
    return false;
}

double vertex::Distance(vertex* fas)
{
    double sqad, root;
    double x,y,z,tx,ty,tz;
    tx = this->x - fas->x;
    ty = this->y - fas->y;
    tz = this->z - fas->z;
    sqad=( (pow(tx,2)) + (pow(ty,2)) + (pow(tz,2)));
    root=sqrt(sqad);
    return root;
}

float vertex::ugol(vertex *sm){
    float a = this->x * sm->x + this->y * sm->y + this->z * sm->z;
    float b = ( sqrt( pow(sm->x,2) + pow(sm->y,2) + pow(sm->z,2) ) * sqrt( pow(this->x,2) +
pow(this->y,2) + pow(this->z,2) ) );
    return a/b;
}
```

Класс STL

```
#ifndef STL_H
#define STL_H

#include <vector>
#include <vertex.h>
#include <table_structure.h>
```

```
#include <QString>
#include <faset.h>

class STL
{
public:
    STL();
    // вес
    double helf = 0.0;
    // ширина
    double width = 0.0;
    // длинна
    double length = 0.0;
    // высота
    double height = 0.0;

    QString name = "";

    QString put = "";

    int idgroup = 0;

    int p = 0;

    // структура
    std::vector<faset*> structure;

    std::vector<faset*> analitic_structure;

    void analisis();

    table_structure* table = new table_structure();

    void save();

    void loadTable(QString table);

    void rodStl();

private:

    int key = 0;

    int vertex_glav(vertex* normals);
    void analis_table_add(int id);
    void analis_faset(faset* fas, int idFaset, int pred_table_normal = 0);
    void smesh_faset(faset* fas, int glav_normal = 0);

    QString strucToStr();

};

#endif // STL_H


#include "stl.h"

#include <QDebug>

#include <QSqlQuery>
#include <QSqlError>

#include <vector>
#include <vertex.h>

#include <line.h>

STL::STL()
{
    delete this->table;
    this->table = new table_structure();
}


void STL::analisis(){
 analitic_structure.clear();
    faset* fas_start;
    for(int i = 0; i < structure.size(); i++){
        fas_start = new faset();
        fas_start->setFaset( this->structure[i] );
```

```
        analitic_structure.push_back( fas_start);
    }
    fas_start = analitic_structure[0];
    fas_start->smesh = true;
    this->analis_faset(fas_start, 0);
    this->rodStl();
}

void STL::rodStl(){

    int F = this->structure.size();

    std::vector<vertex*>p;
    bool vN1, vN2, vN3;
    faset* fas;
    for(int i = 0; i < F; i++){
        vN1 = true;
        vN2 = true;
        vN3 = true;
        fas = this->structure[i];
        for(int j = 0; j < p.size(); j++){
            if ( (p[j]->x == fas->v1->x) &&
                     (p[j]->y == fas->v1->y) &&
                     (p[j]->z == fas->v1->z) ){
                vN1 = false;
            }
            if ( (p[j]->x == fas->v2->x) &&
                     (p[j]->y == fas->v2->y) &&
                     (p[j]->z == fas->v2->z) ){
                vN2 = false;
            }
            if( (p[j]->x == fas->v3->x) &&
                     (p[j]->y == fas->v3->y) &&
                     (p[j]->z == fas->v3->z) ){
                vN3 = false;
            }
        }
        if(vN1)
            p.push_back(fas->v1);

        if(vN2)
            p.push_back(fas->v2);

        if(vN3)
            p.push_back(fas->v3);
    }

    std::vector< Line* > v;

    Line *vrem; // создание динамического массива вещественных чисел на десять элементов

    for(int i = 0; i < F; i++){
        fas = this->structure[i];
        vN1 = true;
        vN2 = true;
        vN3 = true;
        fas = this->structure[i];
        for(int j = 0; j < v.size(); j++){
            // проверка для v1 v2
            if ( ((v[j]->v1.x == fas->v1->x) &&
                     (v[j]->v1.y == fas->v1->y) &&
                     (v[j]->v1.z == fas->v1->z)) &&
                  ((v[j]->v2.x == fas->v2->x) &&
                   (v[j]->v2.y == fas->v2->y) &&
                   (v[j]->v2.z == fas->v2->z))){
                vN1 = false;
            }

            if ( ((v[j]->v2.x == fas->v1->x) &&
                     (v[j]->v2.y == fas->v1->y) &&
                     (v[j]->v2.z == fas->v1->z)) &&
                  ((v[j]->v1.x == fas->v2->x) &&
                   (v[j]->v1.y == fas->v2->y) &&
                   (v[j]->v1.z == fas->v2->z))){
                vN1 = false;
            }

            // проверка для v1 v3
            if ( ((v[j]->v1.x == fas->v1->x) &&
```

```
                    (v[j]->v1.y == fas->v1->y) &&
                    (v[j]->v1.z == fas->v1->z)) &&
                ((v[j]->v2.x == fas->v3->x) &&
                 (v[j]->v2.y == fas->v3->y) &&
                 (v[j]->v2.z == fas->v3->z))){
                vN2 = false;
            }

            if ( ((v[j]->v2.x == fas->v1->x) &&
                    (v[j]->v2.y == fas->v1->y) &&
                    (v[j]->v2.z == fas->v1->z)) &&
                ((v[j]->v1.x == fas->v3->x) &&
                 (v[j]->v1.y == fas->v3->y) &&
                 (v[j]->v1.z == fas->v3->z))){
                vN2 = false;
            }

            // проверка для v2 v3
            if ( ((v[j]->v1.x == fas->v2->x) &&
                    (v[j]->v1.y == fas->v2->y) &&
                    (v[j]->v1.z == fas->v2->z)) &&
                ((v[j]->v2.x == fas->v3->x) &&
                 (v[j]->v2.y == fas->v3->y) &&
                 (v[j]->v2.z == fas->v3->z))){
                vN3 = false;
            }

            if ( ((v[j]->v2.x == fas->v2->x) &&
                    (v[j]->v2.y == fas->v2->y) &&
                    (v[j]->v2.z == fas->v2->z)) &&
                ((v[j]->v1.x == fas->v3->x) &&
                 (v[j]->v1.y == fas->v3->y) &&
                 (v[j]->v1.z == fas->v3->z))){
                vN3 = false;
            }
        }
        if(vN1){
            vrem = new Line;
            vrem->v1 = *fas->v1;
            vrem->v2 = *fas->v2;
            v.push_back(vrem);
        }

        if(vN2){
            vrem = new Line;
            vrem->v1 = *fas->v1;
            vrem->v2 = *fas->v3;
            v.push_back(vrem);
        }

        if(vN3){
            vrem = new Line;
            vrem->v1 = *fas->v2;
            vrem->v2 = *fas->v3;
            v.push_back(vrem);
        }
    }

    this->p = (2 - ( p.size() - v.size() + this->structure.size()) )/2;

    qDebug() << this->p;
}

void STL::analis_faset(faset *fas, int idFaset, int pred_table_normal){
    // определяем к какой части трехмерной таблицы относиться наша нормально
    int id_table_normal = this->vertex_glav(fas->n);
    // помечаем наш фасет как уже обработанный
    fas->glav = true;
    // проверяем, перешли ли мы на новую часть таблицы
    if(pred_table_normal != id_table_normal){
        // если перешли, то увеличиваем номер в таблице на 1
        this->analis_table_add(id_table_normal);
    }
    this->structure[idFaset]->color = this->table->getColorNormal(id_table_normal);

    // находим все смежные фасеты к текущему.
    this->smesh_faset( fas, id_table_normal);

    faset* smesh;
```

```
    bool onePloscals = false;

    for(int i = 0; i < analitic_structure.size(); i++){
        smesh = analitic_structure[i];
        idFaset = i;
        if( smesh->smesh == true && smesh->glav == false){
            onePloscals = true;
            break;
        }
    }

    if(onePloscals){
        this->analis_faset( smesh, idFaset, id_table_normal);
    } else {
        onePloscals = false;

        for(int i = 0; i < analitic_structure.size(); i++){
            smesh = analitic_structure[i];
            idFaset = i;
            if( smesh->smesh == false && smesh->glav == false){
                onePloscals = true;
                break;
            }
        }

        if(onePloscals)
            this->analis_faset( smesh, idFaset, id_table_normal);
    }
}

int STL::vertex_glav(vertex *normals){

    float max_ugol = -9999.9;
    int number_normal = 0;
    float ugol = 0.0;
    for(int i = 1; i < 26; i++){
        ugol = normals->ugol(table->getNormal(i));
        if(ugol > max_ugol ){
            max_ugol = ugol;
            number_normal = i;
        }
    }
    return number_normal;
}

void STL::smesh_faset(faset* fas, int glav_normal){
    faset* smesh;
    for(int i = 0; i < analitic_structure.size(); i++){
        smesh = analitic_structure[i];
        if( fas->smesh_faset( smesh ) && vertex_glav(smesh->n) == glav_normal){
            smesh->smesh = true;
        }
    }
}

void STL::save(){
    if(this->key == 0){
        QSqlQuery query;

        QString str = this->strucToStr();

            query.prepare("INSERT INTO `diplom`.`file` (`name`, `put`, `width`, `length`, `height`,
`structure`, `p`, `idgroup`) "
                          "VALUES ( :name, :put, :width, :length, :heigh, :structure, :p,
:idgroup)");

         query.bindValue( ":name", this->name);
         query.bindValue( ":put", this->put);
         query.bindValue( ":width", this->width);
         query.bindValue( ":length", this->length);
         query.bindValue( ":heigh", this->height);
         query.bindValue( ":structure", str);
         query.bindValue( ":p", this->p);
         query.bindValue( ":idgroup", this->idgroup);

         if( !query.exec() ){
             qDebug() << "> Query exec() error." << query.lastError().text();
         }
```

```
         else {
             qDebug() << ">Query exec() success.";

         }
        int key = query.lastInsertId().toInt();
    }
}
QString STL::strucToStr(){
    QString response = "";
    for(int i = 0; i < 3; i++)
        for(int j = 0; j < 3; j++)
            response+= QString::number(this->table->table[i][j][0]) + "|" + QString::number(this-
>table->table[i][j][1]) + "|" + QString::number(this->table->table[i][j][2]) +  "|";
    return response;
}

void STL::loadTable(QString table){
    QStringList list = table.split("|");

    int x = 0;
    int y = 0;
    int z = 0;

    foreach(QString num, list){
        this->table->table[x][y][z] = num.toInt();
        z++;
        if(z == 3 ){
            z = 0;
            y++;
        }
        if(y == 3 ){
            y = 0;
            x++;
        }
    }
}

void STL::analis_table_add(int id){
    switch (id) {
    case 1:
        this->table->table[1][0][0]++;
        break;
    case 2:
        this->table->table[2][0][0]++;
        break;
    case 3:
        this->table->table[0][1][0]++;
        break;
    case 4:
        this->table->table[0][2][0]++;
        break;
    case 5:
        this->table->table[0][0][1]++;
        break;
    case 6:
        this->table->table[0][0][2]++;
        break;
    case 7:
        this->table->table[1][1][0]++;
        break;
    case 8:
        this->table->table[1][2][0]++;
        break;
    case 9:
        this->table->table[2][1][0]++;
        break;
    case 10:
        this->table->table[2][2][0]++;
        break;
    case 11:
        this->table->table[0][1][1]++;
        break;
    case 12:
        this->table->table[0][1][2]++;
        break;
    case 13:
        this->table->table[0][2][1]++;
        break;
    case 14:
```

```
        this->table->table[0][2][2]++;
        break;
    case 15:
        this->table->table[1][0][1]++;
        break;
    case 16:
        this->table->table[1][0][2]++;
        break;
    case 17:
        this->table->table[2][0][1]++;
        break;
    case 18:
        this->table->table[2][0][2]++;
        break;
    case 19:
        this->table->table[1][1][1]++;
        break;
    case 20:
        this->table->table[1][1][2]++;
        break;
    case 21:
        this->table->table[1][2][1]++;
        break;
    case 22:
        this->table->table[1][2][2]++;
        break;
    case 23:
        this->table->table[2][1][1]++;
        break;
    case 24:
        this->table->table[2][1][2]++;
        break;
    case 25:
        this->table->table[2][2][1]++;
        break;
    case 26:
        this->table->table[2][2][2]++;
        break;
    }
}
```

Класс table_structure

```
#ifndef TABLE_STRUCTURE_H
#define TABLE_STRUCTURE_H

#include <vertex.h>

class table_structure
{
public:
    table_structure();
    int table[3][3][3];

    // id = 1
    vertex* xR = new vertex( 1.0, 0.0, 0.0 ) ;
    // id = 2
    vertex* xL = new vertex( -1.0, 0.0, 0.0 ) ;
    // id = 3
    vertex* yR = new vertex( 0, 1, 0 ) ;
    // id = 4
    vertex* yL = new vertex( 0, -1, 0 ) ;
    // id = 5
    vertex* zR = new vertex( 0, 0, 1 ) ;
    // id = 6
    vertex* zL = new vertex( 0, 0, -1 ) ;
    // id = 7
    vertex* xPyP = new vertex( 1, 1, 0 ) ;
    // id = 8
    vertex* xPyO = new vertex( 1, -1, 0 ) ;
    // id = 9
    vertex* xOyP = new vertex( -1, 1, 0 ) ;
    // id = 10
    vertex* xOyO = new vertex( -1, -1, 0 ) ;
    // id = 11
    vertex* yPzP = new vertex( 0, 1, 1 ) ;
    // id = 12
    vertex* yPzO = new vertex( 0, 1, -1 ) ;
    // id = 13
```

```
    vertex* yOzP = new vertex( 0, -1, 1 ) ;
    // id = 14
    vertex* yOzO = new vertex( 0, -1, -1 ) ;
    // id = 15
    vertex* xPzP = new vertex( 1, 0, 1 ) ;
    // id = 16
    vertex* xPzO = new vertex( 1, 0, -1 ) ;
    // id = 17
    vertex* xOzP = new vertex( -1, 0, 1 ) ;
    // id = 18
    vertex* xOzO = new vertex( -1, 0, -1 ) ;
    // id = 19
    vertex* xPyPzP = new vertex( 1, 1, 1 ) ;
    // id = 20
    vertex* xPyPzO = new vertex( 1, 1, -1 ) ;
    // id = 21
    vertex* xPyOzP = new vertex( 1, -1, 1 ) ;
    // id = 22
    vertex* xPyOzO = new vertex( 1, -1, -1 ) ;
    // id = 23
    vertex* xOyPzP = new vertex( -1, 1, 1 ) ;
    // id = 24
    vertex* xOyPzO = new vertex( -1, 1, -1 ) ;
    // id = 25
    vertex* xOyOzP = new vertex( -1, -1, 1 ) ;
    // id = 26
    vertex* xOyOzO = new vertex( -1, -1, -1 ) ;

    vertex* getNormal(int id);

    vertex getColorNormal(int id);

    int getTableValue(int id);
};

#endif // TABLE_STRUCTURE_H


#include "table_structure.h"
#include <QDebug>

table_structure::table_structure()
{
    for(int i = 0; i < 3; i++)
        for(int j = 0; j < 3; j++)
            for(int z = 0; z < 3; z++)
                this->table[i][j][z] = 0;
}

vertex* table_structure::getNormal(int id){
    switch (id) {
    case 1:
        return this->xR;
        break;
    case 2:
        return this->xL;
        break;
    case 3:
        return this->yR;
        break;
    case 4:
        return this->yL;
        break;
    case 5:
        return this->zR;
        break;
    case 6:
        return this->zL;
        break;
    case 7:
        return this->xPyP;
        break;
    case 8:
        return this->xPyO;
        break;
    case 9:
        return this->xOyP;
        break;
    case 10:
        return this->xOyO;
```

```
        break;
    case 11:
        return this->yPzP;
        break;
    case 12:
        return this->yPzO;
        break;
    case 13:
        return this->yOzP;
        break;
    case 14:
        return this->yOzO;
        break;
    case 15:
        return this->xPzP;
        break;
    case 16:
        return this->xPzO;
        break;
    case 17:
        return this->xOzP;
        break;
    case 18:
        return this->xOzO;
        break;
    case 19:
        return this->xPyPzP;
        break;
    case 20:
        return this->xPyPzO;
        break;
    case 21:
        return this->xPyOzP;
        break;
    case 22:
        return this->xPyOzO;
        break;
    case 23:
        return this->xOyPzP;
        break;
    case 24:
        return this->xOyPzO;
        break;
    case 25:
        return this->xOyOzP;
        break;
    case 26:
        return this->xOyOzO;
        break;
    }
}

vertex table_structure::getColorNormal(int id){
    switch (id) {
    case 1:
        return vertex(1,0,0);
        break;
    case 2:
        return vertex(0.5,0,0);
        break;
    case 3:
        return vertex(0,1,0);
        break;
    case 4:
        return vertex(0,0.5,0);
        break;
    case 5:
        return vertex(0,0,1);
        break;
    case 6:
        return vertex(0,0,0.5);
        break;
    case 7:
        return vertex(1,1,0);
        break;
    case 8:
        return vertex(1,0.5,0);
        break;
    case 9:
```

```
        return vertex(0.5,1,0);
        break;
    case 10:
        return vertex(0.5,0.5,0);
        break;
    case 11:
        return vertex(0,1,1);
        break;
    case 12:
        return vertex(0,1,0.5);
        break;
    case 13:
        return vertex(0,0.5,1);
        break;
    case 14:
        return vertex(0,0.5,0.5);
        break;
    case 15:
        return vertex(1,0,1);
        break;
    case 16:
        return vertex(1,0,0.5);
        break;
    case 17:
        return vertex(0.5,0,1);
        break;
    case 18:
        return vertex(0.5,0,0.5);
        break;
    case 19:
        return vertex(1,1,1);
        break;
    case 20:
        return vertex(1,1,0.5);
        break;
    case 21:
        return vertex(1,0.5,1);
        break;
    case 22:
        return vertex(1,0.5,0.5);
        break;
    case 23:
        return vertex(0.5,1,1);
        break;
    case 24:
        return vertex(0.5,1,0.5);
        break;
    case 25:
        return vertex(0.5,0.5,1);
        break;
    case 26:
        return vertex(0.5,0.5,0.5);
        break;
    }
}

int table_structure::getTableValue(int id){
    switch (id) {
    case 1:
        return this->table[1][0][0];
        break;
    case 2:
        return this->table[2][0][0];
        break;
    case 3:
        return this->table[0][1][0];
        break;
    case 4:
        return this->table[0][2][0];
        break;
    case 5:
        return this->table[0][0][1];
        break;
    case 6:
        return this->table[0][0][2];
        break;
    case 7:
        return this->table[1][1][0];
        break;
```

```
    case 8:
        return this->table[1][2][0];
        break;
    case 9:
        return this->table[2][1][0];
        break;
    case 10:
        return this->table[2][2][0];
        break;
    case 11:
        return this->table[0][1][1];
        break;
    case 12:
        return this->table[0][1][2];
        break;
    case 13:
        return this->table[0][2][1];
        break;
    case 14:
        return this->table[0][2][2];
        break;
    case 15:
        return this->table[1][0][1];
        break;
    case 16:
        return this->table[1][0][2];
        break;
    case 17:
        return this->table[2][0][1];
        break;
    case 18:
        return this->table[2][0][2];
        break;
    case 19:
        return this->table[1][1][1];
        break;
    case 20:
        return this->table[1][1][2];
        break;
    case 21:
        return this->table[1][2][1];
        break;
    case 22:
        return this->table[1][2][2];
        break;
    case 23:
        return this->table[2][1][1];
        break;
    case 24:
        return this->table[2][1][2];
        break;
    case 25:
        return this->table[2][2][1];
        break;
    case 26:
        return this->table[2][2][2];
        break;
    }
}
```