**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ: РОБОТОТЕХНИКА И КОМПЛЕКСНАЯ АВТОМАТИЗАЦИЯ

КАФЕДРА: КОМПЬЮТЕРНЫЕ СИСТЕМЫ АВТОМАТИЗАЦИИ ПРОИЗВОДСТВА

*РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА*

*К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:*

***Булевы операции над полигональными моделями***

Студент РК9-72 **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.А. Кобзарь**

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.С. Хрыков**

*Москва 2016*

Аннотация

Темой работы является «Булевы операции над полигональными моделями». Объем курсовой работы 38 страниц. В ней размещены 37 рисунков и 1 таблица. При написании работы использовалось 6 источников литературы.

Объектом исследования при написании работы послужили полигональные модели. Предметом исследования булевы операции над полигональными моделями.

В курсовую работу входят четыре раздела. В первом разделе описываются существующие методы решения данной задачи, второй и третий разделы посвящены разработке и реализации, а четвертый - тестированию.

Во введении формулируется постановка задачи, описывается область применения.

Заключение посвящено основным выводам и идеям, о дальнейшем развитии темы, поднятой в данной работе.

**Содержание**

[**Введение** 4](#_Toc454827814)

[**Техническое задание**](#_Toc454827817) 6

[**Раздел 1. Литературный обзор**](#_Toc454827820) 9

1.1 Существующие алгоритмы выполнения булевых операций 9

1.2 Выбор алгоритма триангуляции 11

**Раздел** [**2.** **Реализация алгоритма 2D триангуляции**](#_Toc454827821) 14

[2.1 Описание алгоритма](#_Toc454827825) 14

[2.2 Основные блоки программы](#_Toc454827825) 16

[2.3 Результаты 2D триангуляции](#_Toc454827825) 22

**Раздел** [**3.** **Разработка и реализация алгоритма 3D триангуляции**](#_Toc454827821) 26

[**Раздел 4. Тестирование**](#_Toc454827824) 28

[4.1 Тестируемые модели](#_Toc454827825) 28

4[.2 Объединение](#_Toc454827825) 29

[4.3 Пересечение](#_Toc454827826) 31

4[.4 Вычитание](#_Toc454827827) 32

4.5 Результаты тестирования 35

[**Заключение**](#_Toc454827830) 37

[**Литература**](#_Toc454827831) 38

**Введение**

Полигональное моделирование – подход к моделированию объектов путем представления или аппроксимации их поверхности с помощью многоугольников. С ним мы сталкиваемся почти каждый день. Это фильмы, анимация и компьютерные игры. Помимо индустрии развлечений, полигональное моделирование нашло широкое применение в медицине, особенно в хирургии. Программное обеспечение, разрабатываемое в ходе курсового проекта, планируется использовать именно в этой области.

Основным объектом в полигональном моделировании является вершина (точка в трехмерном пространстве). Две вершины, соединенные прямой линией образуют ребро. Три вершины, соединенные друг с другом с помощью ребер, образуют треугольник. Четырехугольники и треугольники являются наиболее распространенными формами, которые используются в полигональном моделировании. Треугольные полигоны предпочтительнее по причине их алгоритмической простоты, численной надежности и эффективности отображения. Для того, чтобы края модели не имели ограненный вид, полигоны должны иметь малый размер.

Группа многоугольников, соединенных общими вершинами называется сеткой.

Существует большое количество операций, которые могут быть выполнены над полигональными сетками. К основным относятся булевы операции:

* пересечение
* вычитание
* объединение

Пересечение – полученный объект является результатом пересечения двух исходных объектов, при этом части объектов, оказавшиеся вне общего внутреннего объема, удаляются.

Вычитание – результатом является объект, полученный посредством вычитания одного объекта из второго, все части которого отсекаются объемом первого и удаляются.

Объединение – результатом операции является объект, который получается вследствие объединения двух исходных объектов, при этом части объектов, оказавшиеся внутри общего внешнего объема, удаляются.

Целью данного курсового проекта является написание программы для выполнения булевых операций над полигональными моделями.

**Техническое задание**

1. **Введение**

Программа предназначена к применению в области полигонального моделирования для осуществления булевых операций, пришедших в 3D - графику из булевой алгебры и основанных на понятиях объединения, пересечения и вычитания.

**2. Назначение разработки**

Функциональные назначение программы состоит в предоставлении пользователю возможности выполнения булевых операций над трёхмерными моделями, представленными полигональными сетками.

**3. Требования к программе**

**3.1 Требования к функциональным характеристикам**

**3.1.1 Требования к составу выполняемых функций**

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

а) функции открытия (загрузки) существующих файлов формата STL

б) функции редактирования открытого файла (выполнение булевых опе-раций) с применением стандартных устройств ввода

в) функции сохранения результата применения булевых операций в ви-де файла формата STL

**3.1.2 Требования к организации входных данных**

Входные данные программы должны быть организованы в виде файлов формата STL. Файлы указанного формата должны размещаться (храниться) на

локальных или съёмных носителях, отформатированных согласно требованиям операционной системы. Любой файл иного формата открываться не должен.

**3.1.3 Требования к организации выходных данных**

Выходные данные программы: файл формата STL.

**3.1.4 Требования к временным характеристикам**

Требования к временным характеристикам программы не предъявляются.

**3.2 Требования к надежности**

Информационная структура входных файлов должна включать в себя текст, содержащий разметку, предусмотренную спецификацией формата STL.

**3.3 Условия эксплуатации**

**3.3.1 Климатические условия эксплуатации**

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

**3.3.2 Требования к составу и параметрам технических средств**

В состав технических средств должен входить персональный компьютер, удовлетворяющий следующим требованиям:

• ОС: Windows XP или новее

• процессор Intel Pentium 4 или лучше

• оперативная память: 256 Мб

• жесткий диск: не менее 100Мб свободного места

**3.4 Требования к информационной и программной совместимости**

**3.4.1 Требования к исходным кодам и языкам программирования**

Исходные коды программы должны быть реализованы на языке С++. В качестве интегрированной среды разработки программы должна быть использована среда Visual Studio.

**3.4.2 Требования к программным средствам, используемым программой**

Системные программные средства, используемые программой, должны быть представлены лицензионной версией операционной системы.

**3.4.3 Требования к защите информации и программ**

Требований к защите информации и программ не предъявляются.

**4. Технико-экономические показатели**

Программа совместима с операционной системой Windows, а значит доступна большому кругу потребителей. На данный момент на рынке не представлено ни одной аналогичной программы российской разработки. Зарубежные аналоги малодоступны и имеют недостатки, которые учитываются при разработке программы.

**5. Стадии и этапы разработки**

**5.1 Стадии разработки**

1) Исследование предметной области

2) Реализация выполнения булевых операций с полигональными моделями

**5.2 Этапы разработки**

1) Подготовка набора тестовых моделей

2) Проектирование и реализация алгоритма полного перебора для поиска линий пересечения моделей

3) Реализация алгоритма триангуляции Делонэ пересекающихся участков моделей

4) Разбиение моделей на внешние и внутренние части с учетом триан-гулированных поверхностей

5) Реализация операций «Объединение», «Вычитание» и «Пересечение»

6) Тестирование программы с использованием реальных моделей (детали, кости)

**Раздел 1. Литературный обзор**

**1.1 Существующие алгоритмы выполнения булевых операций**

Основная проблема булевых операций заключается в том, что результатом некоторой операции даже над двумя простыми выпуклыми многоугольниками в общем случае может являться множество невыпуклых многоугольников с возможными отверстиями и самокасаниями. К сожалению, традиционные алгоритмы ограничиваются тем, что разбивают сложные многоугольники результата на простые, тем самым исключая возможность повторного выполнения операций над результатом. Для задач отсечения по экрану этого достаточно, но в моделировании в целях медицины (имплантаты) требуется реализация замкнутого набора операций над множествами многоугольников. Это означает, что результат, полученный с помощью алгоритма, должен удовлетворять его же предусловию. Также желательно, чтобы алгоритм допускал отверстия и неограниченную кратность вершин. Проведём краткий обзор существующих алгоритмов.

Вейлер (Weiler) и Азертон (Atherton) [1] предложили следующий алгоритм. Ребра исходных многоугольников дублируются, и между ними вычисляются пересечения. Затем производится перераспределение ссылок в вершинах пересекающихся ребер с целью выделения минимальных контуров. Для этого используется довольно сложный набор правил для различных случаев пересечения рёбер. На основе полученных контуров создаются результирующие многоугольники для операций объединения, пересечения и вычитания.

Перечислим основные недостатки этого подхода:

— результат булевой операции не всегда состоит из минимальных контуров;

— не освещена проблема описания многоугольников с самокасаниями, которые могут быть получены в результате работы алгоритма;

— результат операции может не удовлетворять требованиям на входные данные, причем расширение алгоритма для обработки кратных пересечений крайне затруднительно.

Шутте (Schutte) модифицировал алгоритм Вейлера более четким разделением этапов и новым алгоритмом маркировки ребер (edge labeling), однако его алгоритм обладает более существенными ограничениями на входные данные. Итак, приведенные выше алгоритмы реализуют набор операций, не являющийся замкнутым, так как в результате их выполнения могут получаться контуры с самокасаниями, что неприемлемо в качестве исходных данных для обоих алгоритмов, и с отверстиями, которые не допускаются алгоритмом Шутте. Сравнительно недавно Гютинг (Gueting) и Шнайдер (Schneider) разработали ROSE-алгебру [2] (RObust Spatial Extension algebra), реализующую замкнутый набор операций над двумерными объектами, в том числе регионами (regions). Предложенное ими описание также имеет ряд недостатков:

— координаты вершин ROSE-объектов принадлежат дискретному множеству, в отличие от общепринятой в вычислительной геометрии модели вещественнозначной РАМ ;

— сложный и громоздкий в реализации алгоритм связывания сегментов (half-segments) региона в корректные внешние и внутренние циклы (R-cycles), приводящий к снижению эффективности процедур обработки регионов.

В результате анализа существующих решений задачи реализации булевых операций над полигональными моделями в виду явных недостатков каждого из них было решено разделить задачу на более мелкие, и отдельно подбирать алгоритмы для решения каждой подзадачи. Подзадачи были выделены таким образом:

1. Проектирование и реализация алгоритма для поиска линий пересечения моделей

2. Реализация алгоритма триангуляции пересекающихся участков моделей

4. Разбиение моделей на внешние и внутренние части с учетом триангулированных поверхностей

5. Реализация операций «Объединение», «Вычитание» и «Пересечение»

**1.2 Выбор алгоритма триангуляции**

После поиска пересечения моделей необходимо выполнить триангуляцию пересекающихся треугольников. Без этого этапа было бы невозможно разбиение моделей на внешние и внутренние части.

К выполнению задачи триангуляции можно подойти по – разному: Например, треугольник (рис. 1) разбивается линией пересечения ABC на два полигона: оранжевый и зеленый. Таким образом, можно выполнить отдельно триангуляцию каждого получившегося полигона, т.е. встает задача о триангуляции многоугольника.

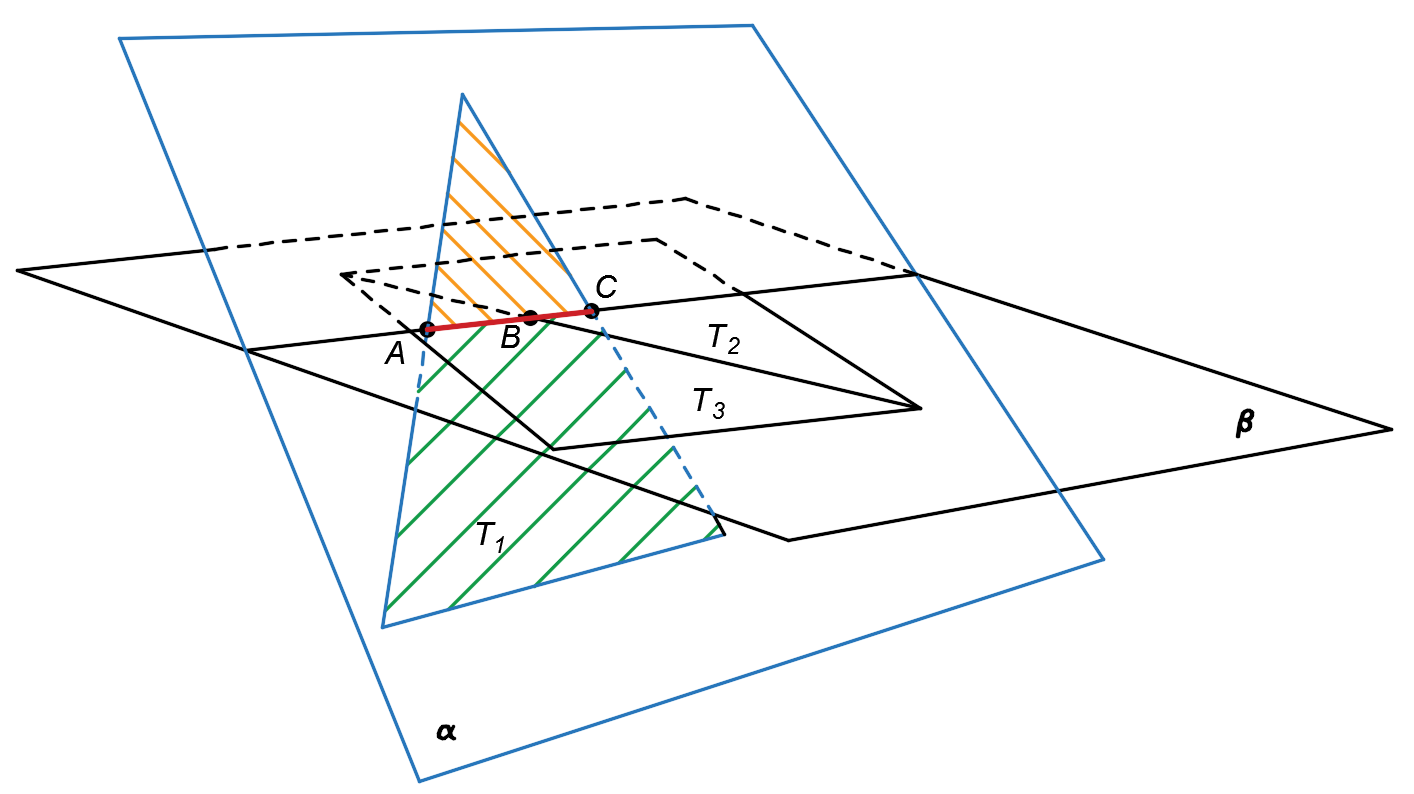


Рис. 1. Пересечение треугольников

Во втором случае мы будем работать не с полигонами, а с точками (рис. 2). В один массиве записываются как точки пересечения, принадлежащие треугольнику, так и вершины этого треугольника. Таким образом, перед нами встает задача построения триангуляции по заданному набору точек. Стоит отметить, что при выполнении такой триангуляции всегда получится такая же кривая пересечения, что и в случае, рассмотренном выше.

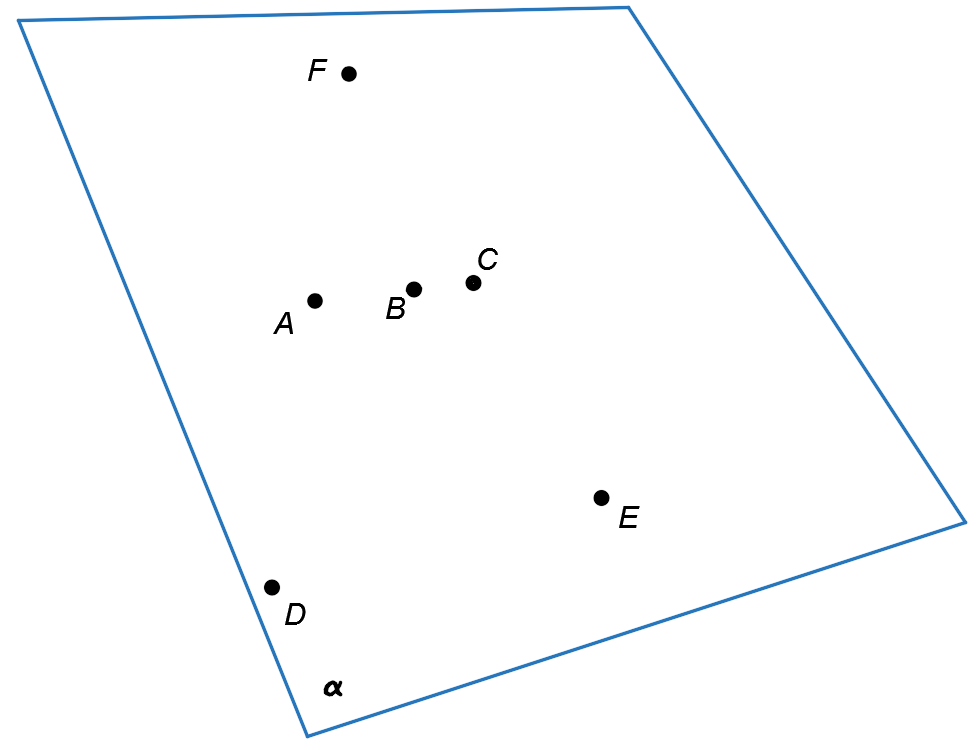
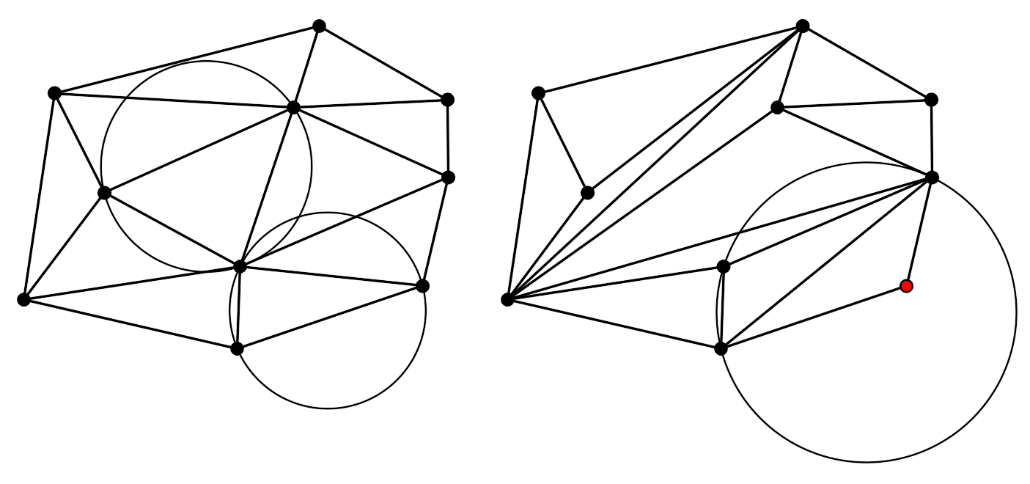


Рис. 2. Набор точек для триангуляции

В данном курсовом проекте была выбрана триангуляция по заданному набору точек. Этот выбор был сделан по причине сложности решения задачи о триангуляции невыпуклого многоугольника.

Для решения задачи построения триангуляции по заданному набору точек была выбрана триангуляция Делоне. Она хорошо сбалансирована в том смысле, что формируемые треугольники стремятся к равноугольности. Так например, триангуляцию, изображенную на рис. 3а, можно отнести к триангуляции Делоне, а на рис. 3б триангуляция содержит несколько сильно вытянутых треугольников и её нельзя отнести к типу Делоне.



а) б)

Рис. 3 Два различных варианта триангуляции для одного набора точек

Триангуляция набора точек будет триангуляцией Делоне, если описанная окружность для каждого треугольника будет свободна от точек. На схеме триангуляции рис.3а показаны две окружности, которые явно не содержат внутри себя других точек (можно провести окружности и для других треугольников, чтобы убедиться, что они также свободны от точек набора). Это правило не соблюдается на схеме рис. 3б – внутрь проведенной окружности попала одна точка (выделана красным) другого треугольника, следовательно, это триангуляция не относится к типу Делоне. На рис. 4. Показан пример триангуляции Делоне для набора большого числа точек.

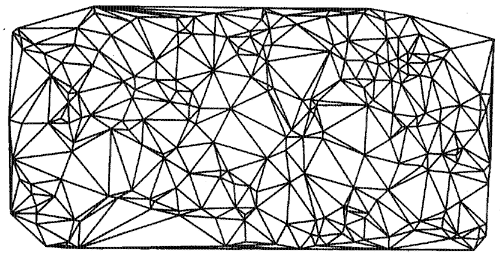


Рис. 4. Триангуляция Делоне для 250 точек.

Существует большое количество алгоритмов построения триангуляции Делоне. Одним из наиболее простых является инкрементальный алгоритм, работающий за O () операций. Именно этот алгоритм был выбран в данном курсовом проекте. Вначале он был реализован для триангуляции точек, лежащих в плоскости X0Y, а затем доработан до полноценного алгоритма, работающего с плоскостями, произвольно ориентированными в пространстве.

**Раздел 2. Реализация алгоритма 2D триангуляции**

**2.1. Описание алгоритма**

Инкрементальный алгоритм работает путем постоянного наращивания текущей триангуляции по одному треугольнику за один шаг. Вначале текущая состоит из единственного ребра оболочки, по окончании работы алгоритма текущая триангуляция становится триангуляцией Делоне. На каждой итерации алгоритм ищет новый треугольник, который подключается к границе текущей триангуляции.

Определение границы зависит от следующей схемы классификации ребер триангуляции Делоне (рис.5) относительно текущей триангуляции. Каждое ребро может быть спящим, живым или мертвым:

* спящие ребра: ребро триангуляции Делоне является спящим, если оно еще не было обнаружено алгоритмом
* живые ребра: ребро живое, если оно обнаружено, но известна только одна примыкающая к нему область
* мертвые ребра: ребро считается мертвым, если оно обнаружено и известны обе примыкающие к нему области

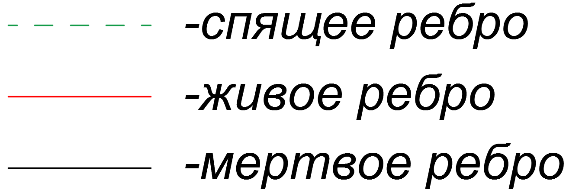
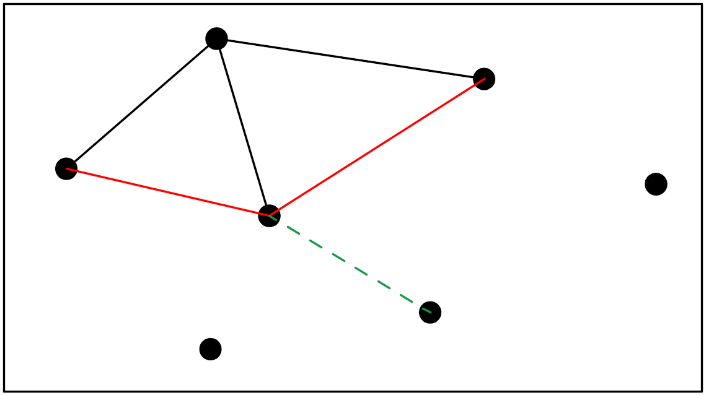


Рис. 5. Классификация ребер

Вначале живым является единственное ребро, принадлежащее выпуклой оболочке – к нему примыкает неограниченная плоскость, а все остальные ребра спящие. По мере работы алгоритма ребра из спящих становятся живыми, затем мертвыми. Граница на каждом этапе состоит из набора живых ребер.

На каждой итерации выбирается наибольшее ребро (для сравнения ребер используется функция cmpEdges (рис.8)) из набора живых ребер *е* границы и оно подвергается обработке, заключающейся в поиске неизвестной области, которой принадлежит ребро *е*. Если эта область окажется треугольником *t,* определяемым концевыми точками ребра *е* и некоторой третьей вершиной *v,* то ребро *е* становится мертвым, поскольку теперь известны обе примыкающие к нему области. Каждое из двух других ребер треугольника *t,* переводятся в следующее состояние: из спящего в живое или из живого в мертвое. Здесь вершина *v* будет называться сопряженной с ребром *е.* В противном случае, если неизвестная область оказывается бесконечной плоскостью, то ребро *е* просто умирает. В этом случае ребро *е* не имеет сопряженной вершины.

На рис. 6 показана работа алгоритма, где действие происходит сверху вниз и слева направо. Граница на каждом этапе показана красной линией.

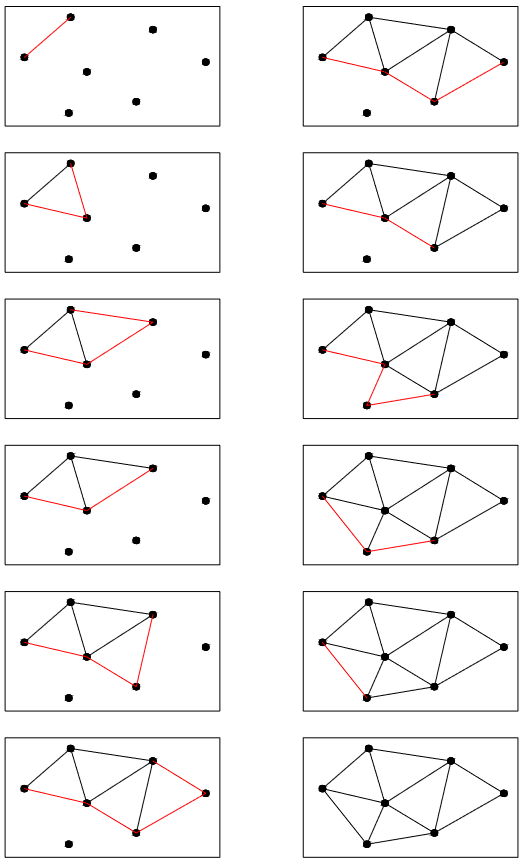


Рис. 6. Наращивание триангуляции Делоне

**2.2 Основные блоки программы**

Алгоритм реализован в функции buildTrianglesUsingIncAlg (рис. 7). Функции задается массив points и она возвращает массив треугольников, представляющие триангуляцию Делоне через ссылочный параметр triangles.

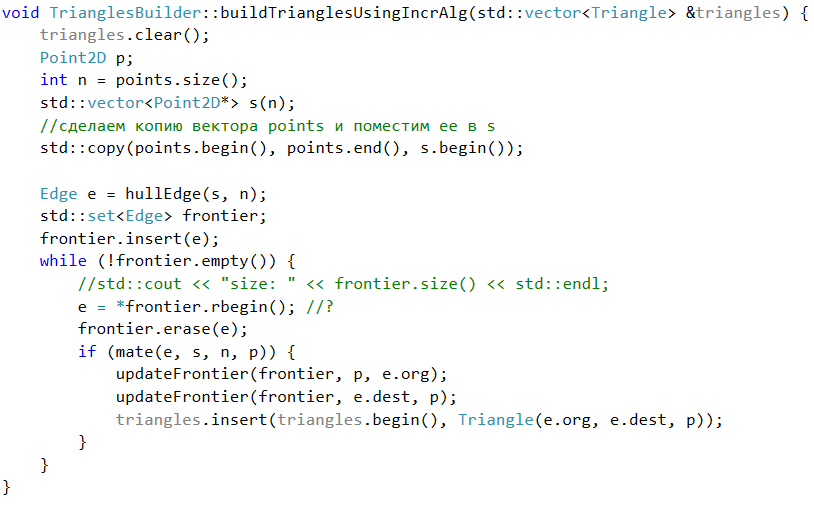


Рис. 7. Функция триангуляции

Треугольники, образующие триангуляцию записываются в список triangles. Граница представлена словарем frontier живых ребер. Каждое ребро направлено так, что неизвестная область для него (подлежащая определению) лежит справа от ребра. Для просмотра словаря используется функция cmpEdges (рис. 8). В ней сравниваются начальные точки двух ребер, если они оказываются равными, то потом сравниваются их концевые точки:

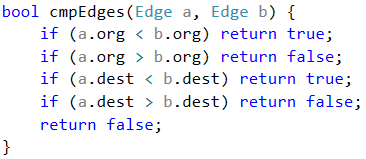


Рис. 8. Функция сравнения ребер

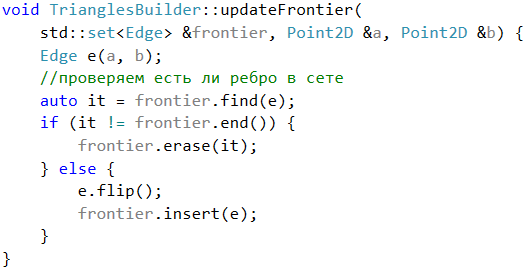
Функция updateFrontier (рис. 9) изменяет словарь ребер границы. 

Рис. 9. Функция изменения словаря

При присоединении к границе нового треугольника t изменяются состояния трех ребер треугольника. Ребро треугольника t, примыкающее к границе, из живого становится мертвым. Функция updateFrontier может игнорировать это ребро, поскольку оно уже должно быть удалено из словаря при обращении к функции erase. Каждое из двух оставшихся ребер изменяет свое состояние из спящего на живое, если они уже ранее не были записаны в словарь, или из живого в мертвое, если ребро уже находится в словаре. На рис. 10 показаны оба случая. В соответствии с рисунком мы обрабатываем живое ребро af и, после обнаружения, что точка b является сопряженной ему, добавляем треугольник afb к текущей триангуляции. Затем ищем ребро fb в словаре и, поскольку его там еще нет и оно обнаружено впервые, его состояние изменяется от спящего к живому. Для редактирования словаря мы повернем ребро fb так, чтобы примыкающая к нему неизвестная область лежала справа от него и запишем это ребро в словарь. Затем отыщем в словаре ребро ba – поскольку оно есть в нем, то оно уже живое (известная примыкающая к нему область – треугольник abc). Так как неизвестная для него область, треугольник afb, только что была обнаружена, это ребро удаляется из словаря.

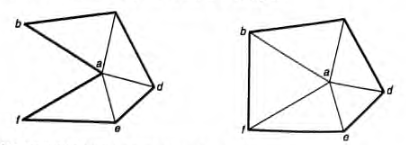


Рис. 10. Подключения треугольника afb к живому ребру af

Функция hullEdge (рис. 11) обнаруживает ребро оболочки среди n точек массива s.

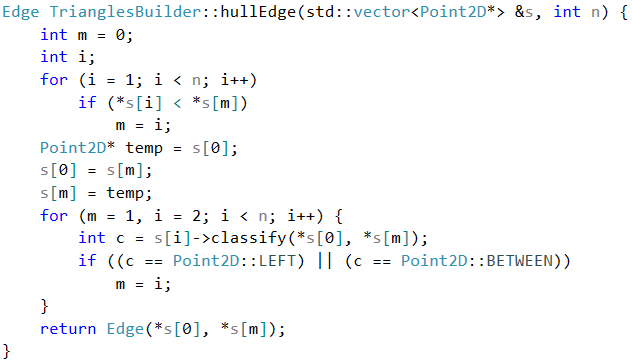


Рис. 11. Функция обнаружения ребра оболочки

Функция mate определяет, существует ли для данного живого ребра сопряженная точка и, если она есть, находит ее. Любое ребро ab определяет бесконечное семейство окружностей, проходящих через его концевые точки a и b. Обозначим это семейство окружностей через С(a,b) (рис. 12).



Рис. 12. Четыре окружности из семейства С(a,b), определяемого ребром ab и их параметрические значения

Центры окружностей семейства C(a,b) лежат на прямой линии, перпендикулярной отрезку ab и проходящей через его центр, и для них можно определить однозначное соотношение с точками на этом перпендикуляре. Для спецификации окружностей семейства параметризуем перпендикуляр – припишем каждой окружности параметрическое значение положения ее центра. На рис. 12 используется запись для обозначения окружности, соответствующей параметрическому значению r.

Рассмотрим поиск сопряженной точки для некоторого живого ребра ab среди множества точек. Предположим, что окружность является описанной окружностью для известной области ребра ab (на рис. 13 такой известной областью будет треугольник abc). Если известная область является неограниченной, то r = -∞, и представляет собой полуплоскость, лежащую слева от ребра ab. Нам нужно найти такой наименьшее значение t > r, чтобы некоторая точка из набора (отличная от a и b), принадлежала окружности . Если не существует такого значения t, то ребро ab не имеет сопряженной точки.

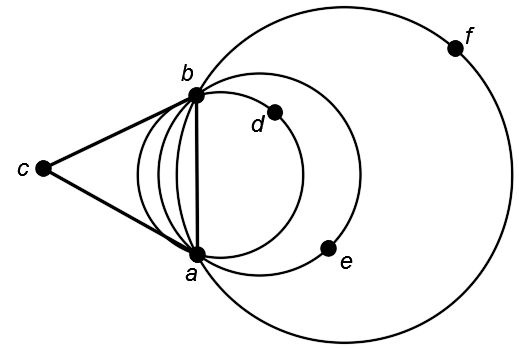


Рис. 13. Определение сопряженной точки ребра ab

Докажем, что такой алгоритм работает. Пусть обозначает описанную окружность известной области отрезка ab и – описанную окружность неизвестной области отрезка ab. Здесь t > r и t = ∞, если отрезок ab не имеет сопряженной точки. Будет ли окружность свободной от точек? Слева от отрезка ab окружность должна быть свободной от точек, так как свободна от точек, а часть , которая лежит слева от ребра ab, находится внутри . Справа от ребра ab также должна быть свободна от точек, поскольку если бы некоторая точка q попала бы внутрь этой окружности, то она бы принадлежала окружности , где r < a < t, что противоречило бы нашему выбору t. Окружность достигла бы точки q до того, как он достиг сопряженной точки ребра ab.

При поиске сопряженной точки для ребра ab рассматриваются только те точки, которые лежат справа от ab. Центр окружности, описанной вокруг любых трех точек a, b, c, лежит на пересечении перпендикуляров, проведенных через середины отрезков ab и bp (перпендикуляры в серединах ребер треугольников пересекаются в центре описанной окружности треугольника). Вместо вычисления положения центра окружности мы вычисляем его параметрическое значение вдоль перпендикуляра к середине ребра ab. Таким образом можно осуществить поиск наименьшего параметрического значения.

Этот алгоритм реализован в функции mate (рис. 14), которая возвращает значение true, если ребро e имеет сопряженную точку, и false, если такой точки нет. Если точка существует, то она возвращается через ссылочный параметр p.

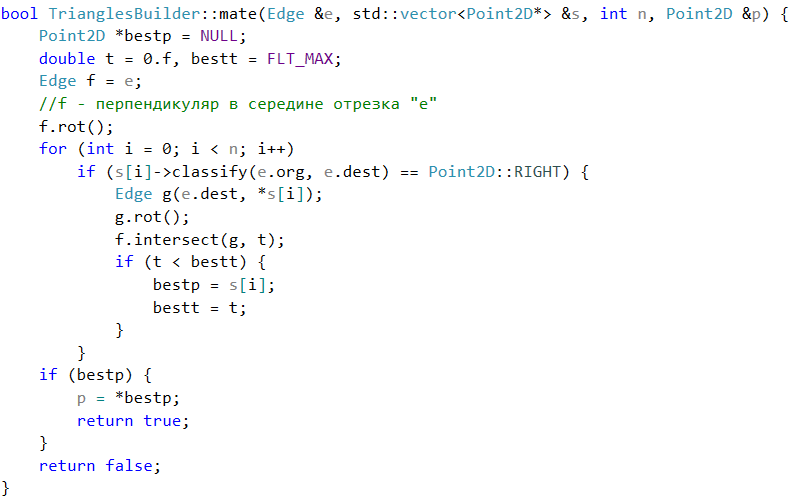


Рис. 14. Функция поиска сопряженной точки

В функции mate переменная bestp указывает на самую лучшую точку, найденную к данному моменту, а в переменной best хранится параметрическое значение для окружности, которое проходит через эту точку.

Этот алгоритм для вычисления триангуляции Делоне по набору из n точек выполняется за время O(), так как при каждой итерации из границы исключается одно ребро. Поскольку каждое ребро исключается из границы точно однажды – каждое ребро относится к границе однажды и затем исключается из нее, никогда не возвращаясь - число итераций равно числу ребер в триангуляции Делоне. Согласно теореме о триангуляции набора точек любая триангуляция содержит не более, чем О(n) ребер, поэтому алгоритм выполняет О(n) итераций. Поскольку на каждую итерацию тратится время О(n), то полностью алгоритм выполняется за время O().

**2.3 Результаты 2D триангуляции**

Инкрементальный алгоритм триангуляции Делоне был реализован. А также при помощи OpenGl было написано приложение для отладки алгоритма. В этом приложении кликом мыши задаются точки, из которых будет получена триангуляция Делоне (рис. 15, 16).

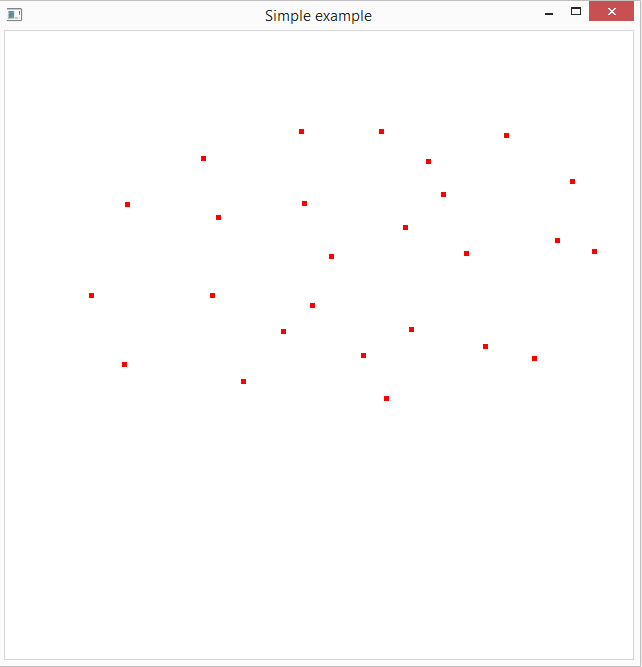


Рис. 15. Исходные точки

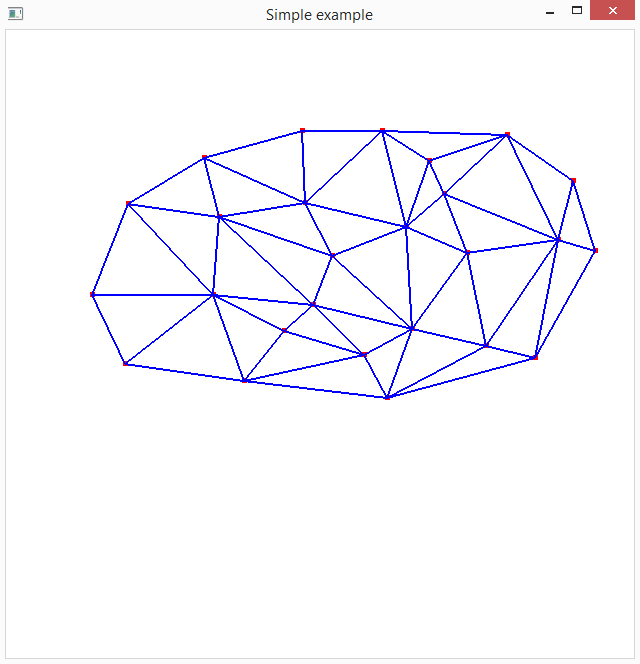


Рис. 16. Результат триангуляции

Возможно добавление точек в уже существующую триангуляцию (рис. 17, 18).

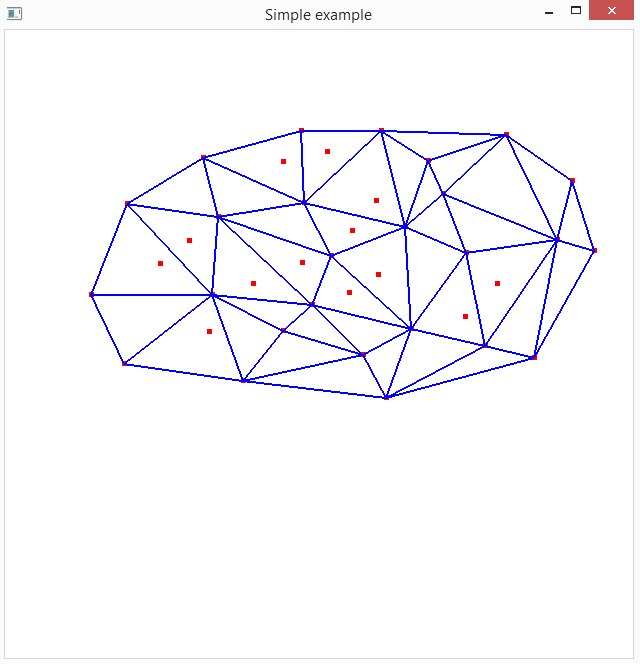


Рис. 17. Добавление точек в существующую триангуляцию

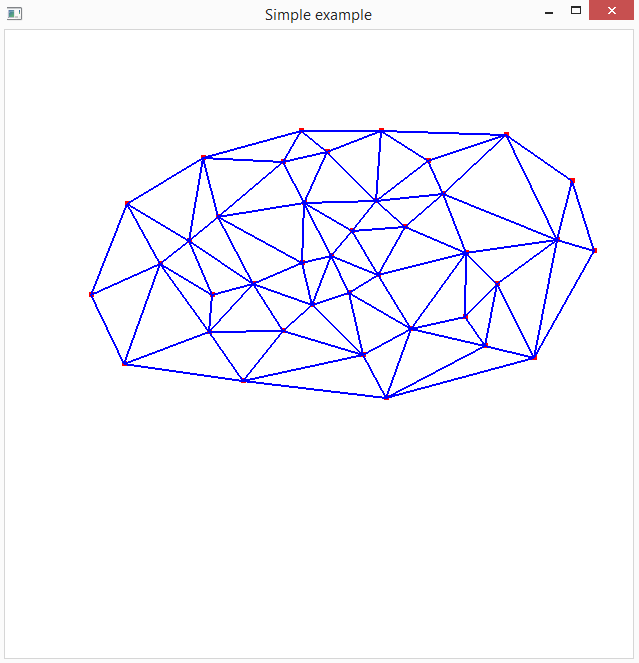


Рис. 18. Результат триангуляции

**Раздел 3. Разработка и реализация алгоритма 3D триангуляции**

В этой главе описываются изменения, внесенные в инкрементальный алгоритм триангуляции Делоне для его работы с точками, лежащими в произвольно ориентированных в пространстве плоскостях.

Основные изменения показаны на рис. 19. Для каждой плоскости вычисляется нормаль. Затем происходит классификация нормали в функции classifyNormal (рис. 20), позволяющая обрабатывать все частные случаи. Следующим шагом выполняется отражение координат (меняем местами координаты точек), сводя тем самым задачу к уже решенной, т.е. временно убирая третью координату, так как все операции над точками будут проводиться с двумя координатами. В алгоритме используется функция поворота ребра вокруг своего центра на 90° по часовой стрелке. Для корректного ее выполнения необходимо обеспечить, чтобы нормаль к плоскости и единичные вектора и образовывали правую тройку векторов.

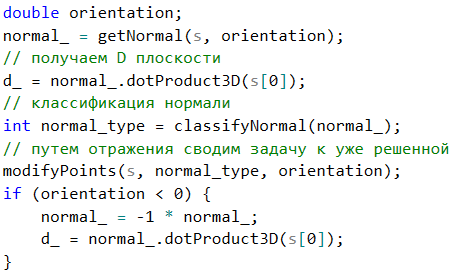


Рис. 19. Изменения в алгоритме

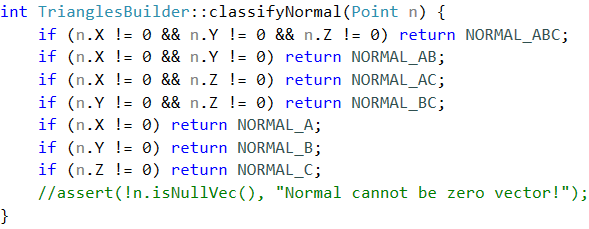


Рис. 20. Классификация нормалей

Рассмотрим пример. На рис. 21 показан случай, когда плоскость параллельна оси Z. Тогда нормаль будет иметь координаты (A, B, 0). Вначале выполняется отражение координат: меняем местами координаты Z и Y для всех точек, принадлежащих плоскости. После выполнения замены изменилась ориентация системы координат: она из правой стала левой, следовательно, необходимо выполнить преобразование, т.е. поменять местами Х и Y. Затем определяется ориентация: если orientation < 0, то необходимо инвертировать нормаль, чтобы она образовывала правую тройку с и .

После нахождения сопряженной точки координаты точек, которые образуют новый треугольник, преобразуются обратно и происходит запись треугольника в массив.

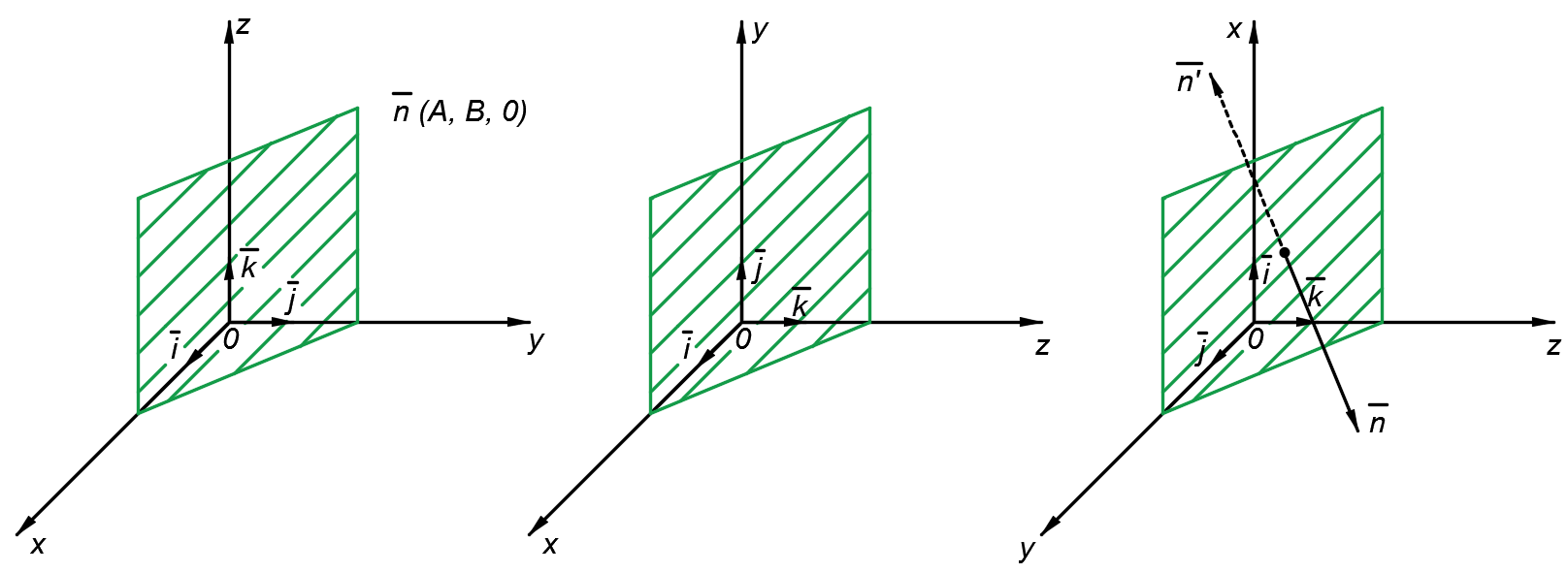


Рис. 21. Преобразование координат

**Раздел 4. Тестирование**

**4.1 Тестируемые модели**

В данном разделе будут рассмотрены результаты применения операций объединения, пересечения и вычитания на наборе тестируемых моделей: конус, сфера, куб, параллелепипед, кость позвоночника и конический винт. Модели были выбраны таким образом, чтобы наиболее наглядно продемонстрировать возможности написанной программы.

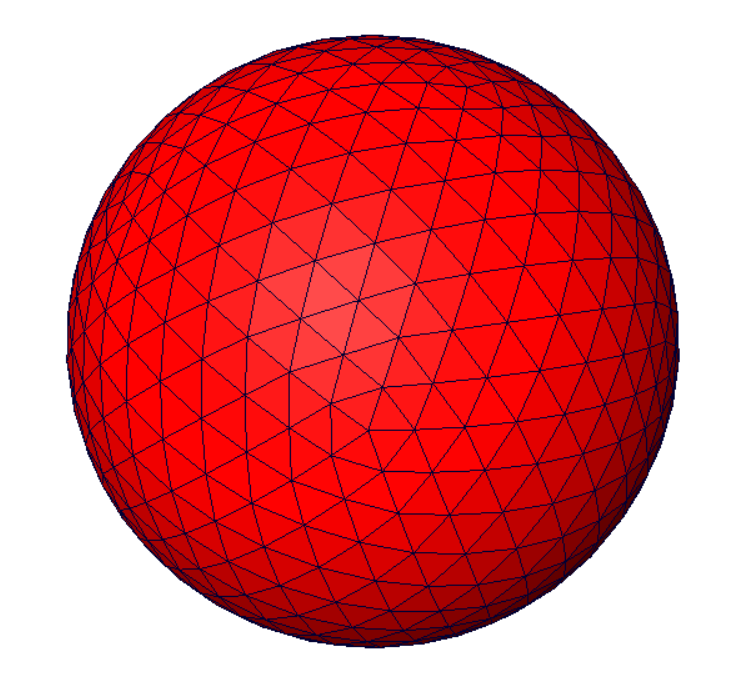


Рис. 22. Конус и сфера

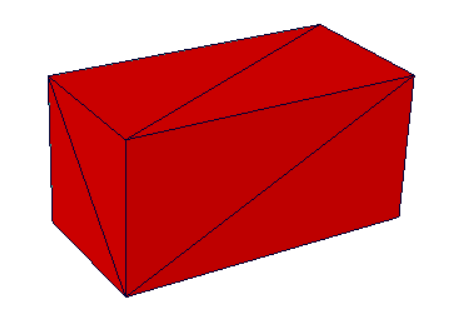
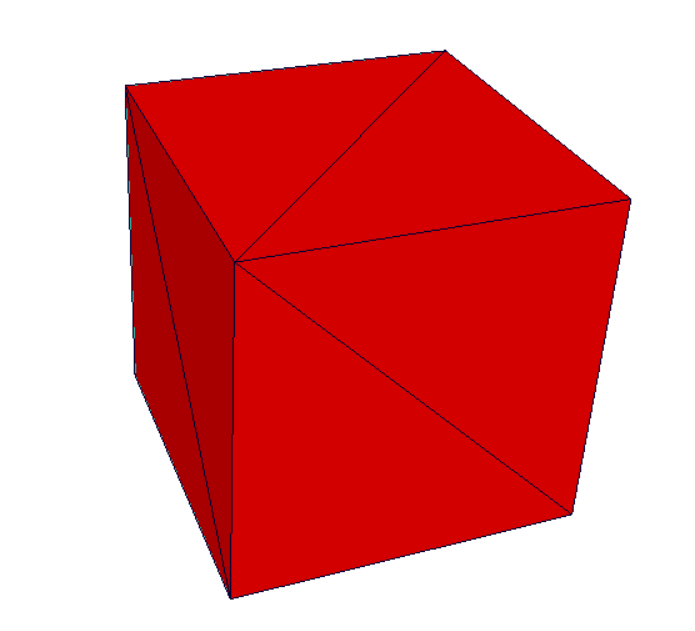


Рис. 23. Куб и параллелепипед

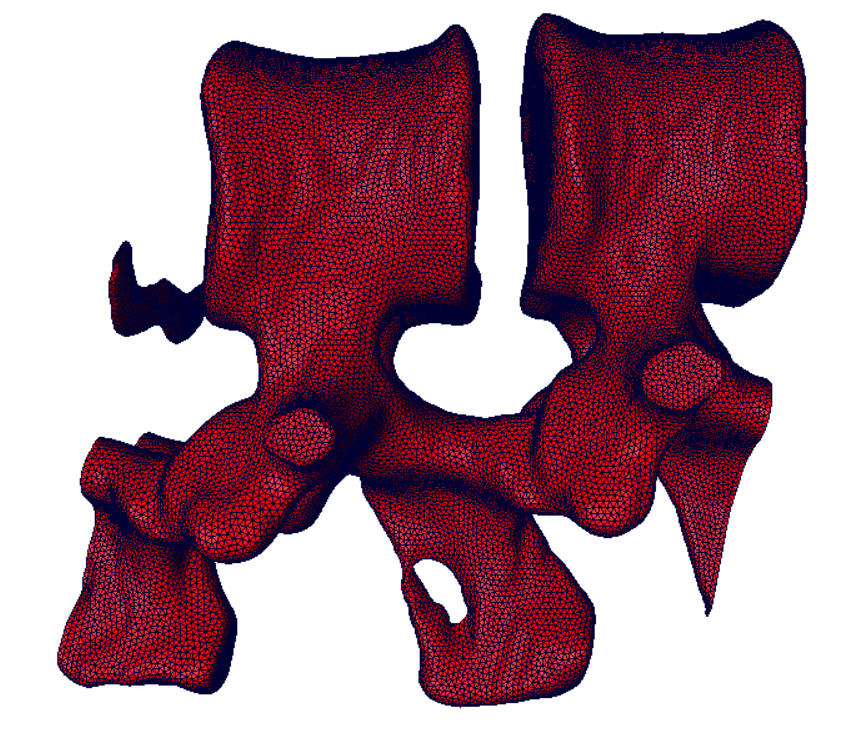
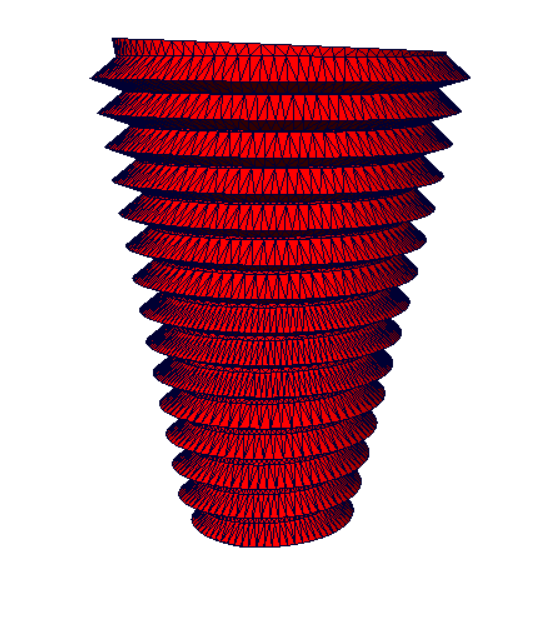
 

Рис. 24. Кость позвоночника и конический винт

**4.2 Объединение**

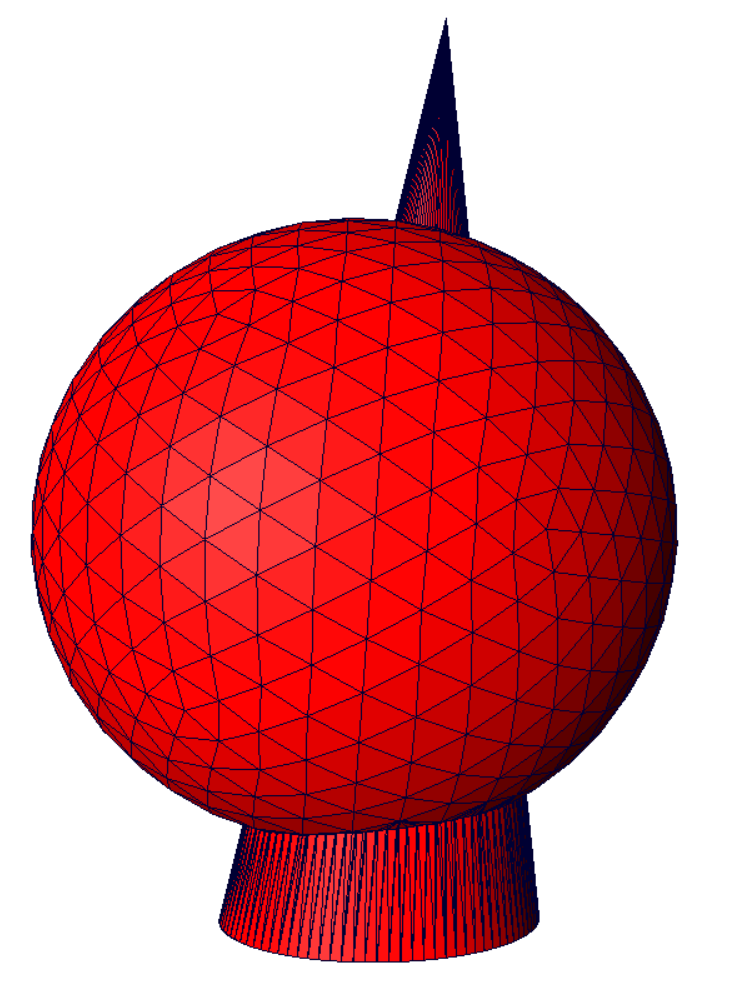


Рис. 25. Результат объединения конуса и сферы

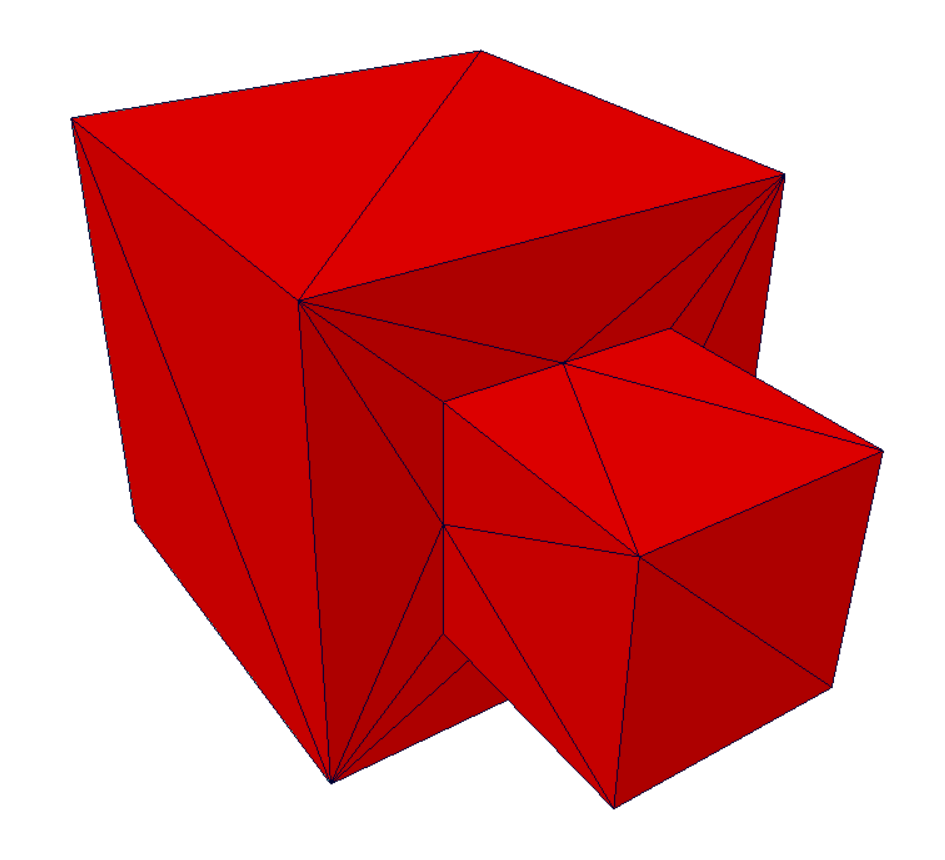


Рис. 26. Результат объединения куба и параллелепипеда

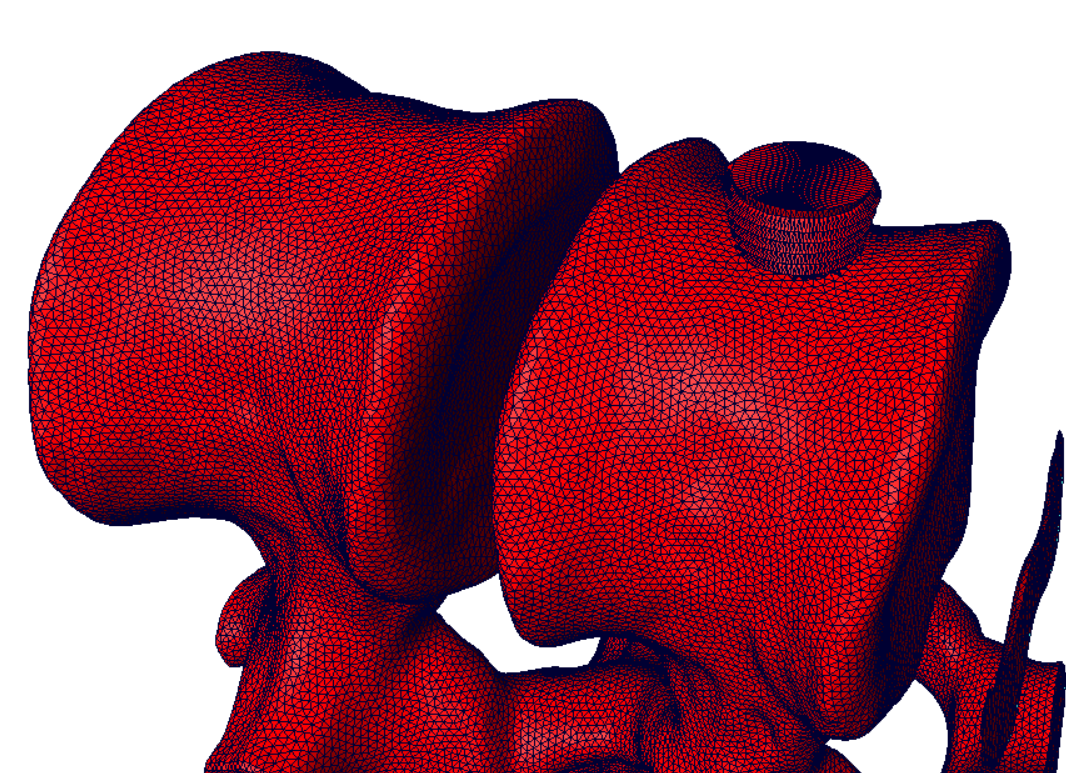


Рис. 27. Результат объединения кости позвоночника и конического винта

**4.3 Пересечение**

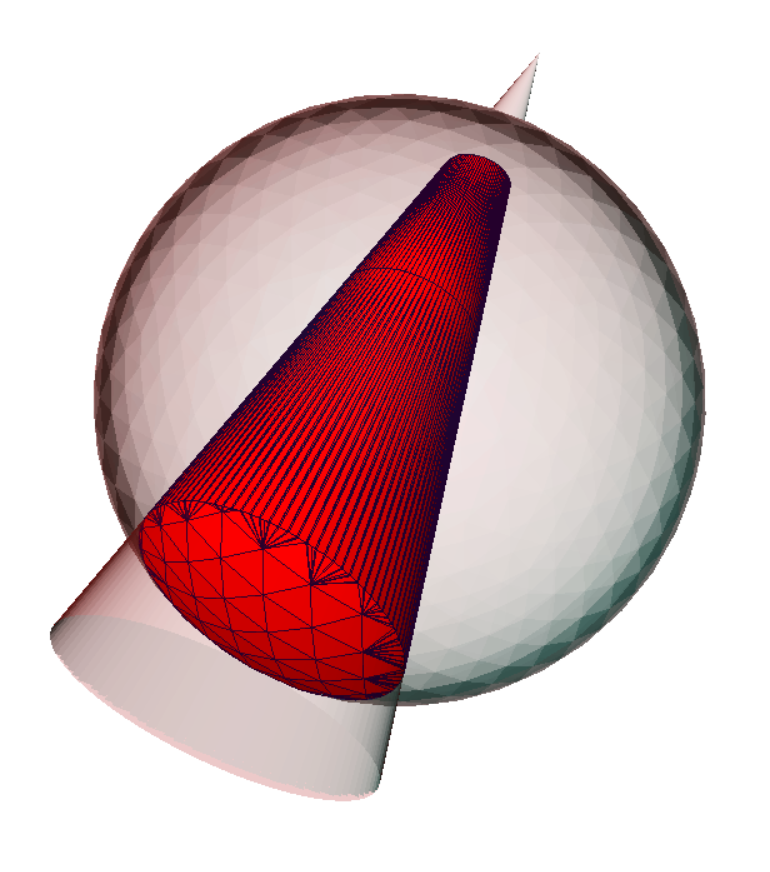


Рис. 28. Результат пересечения конуса и сферы

(прозрачные модели – исходные)

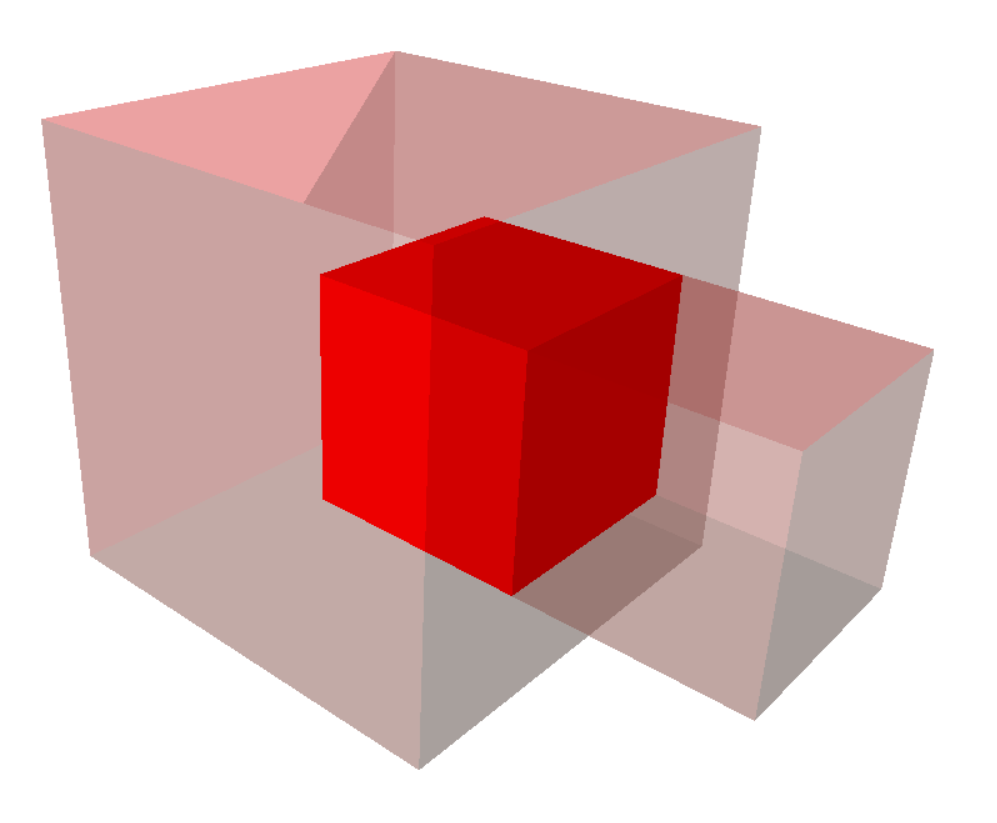


Рис. 29. Результат пересечения куба и параллелепипеда

(прозрачные модели – исходные)

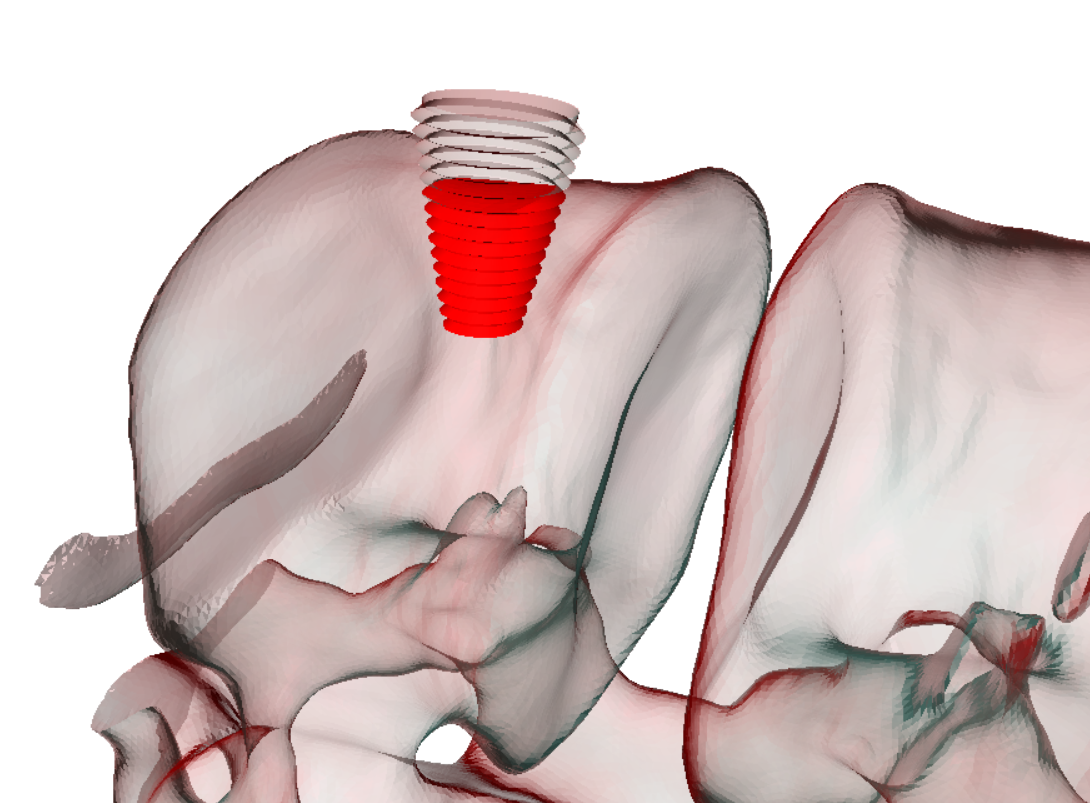


Рис. 30. Результат пересечения кости позвоночника и конического винта

(прозрачные модели – исходные)

**4.4 Вычитание**

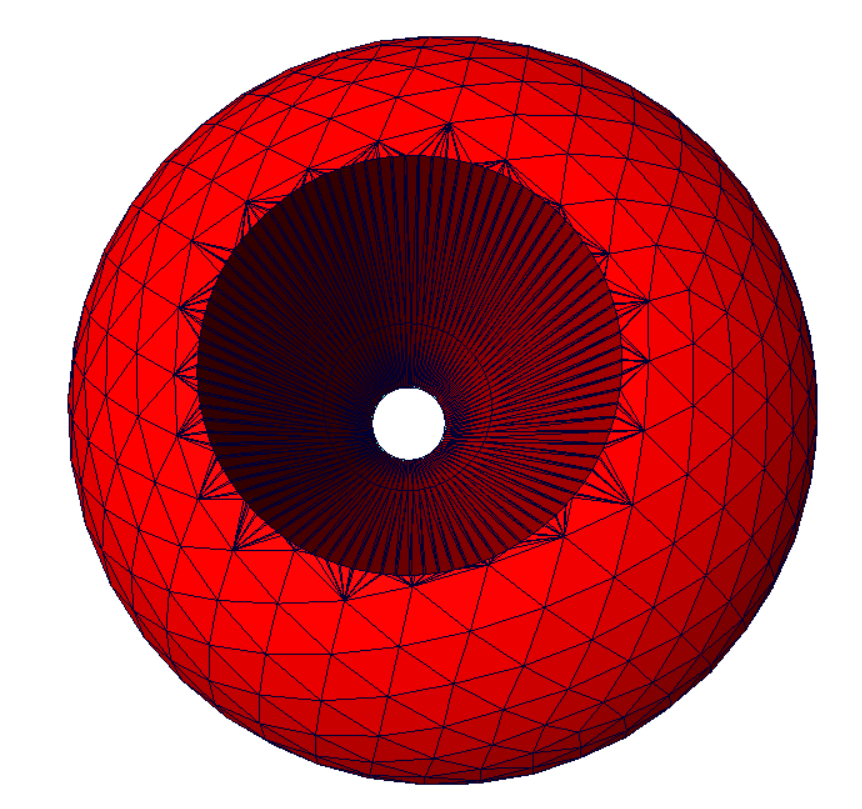


Рис. 31. Результат вычитания из сферы конуса

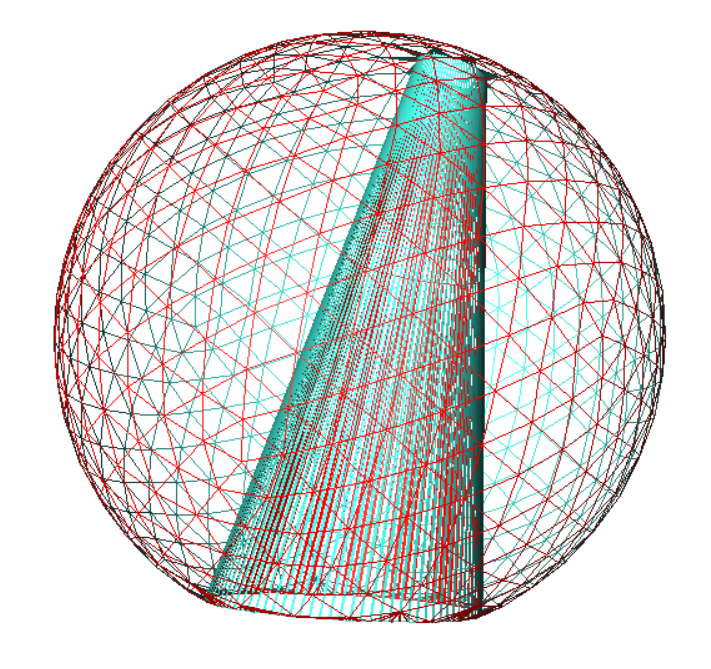


Рис. 32. Результат вычитания конуса из сферы (полигональная сетка)

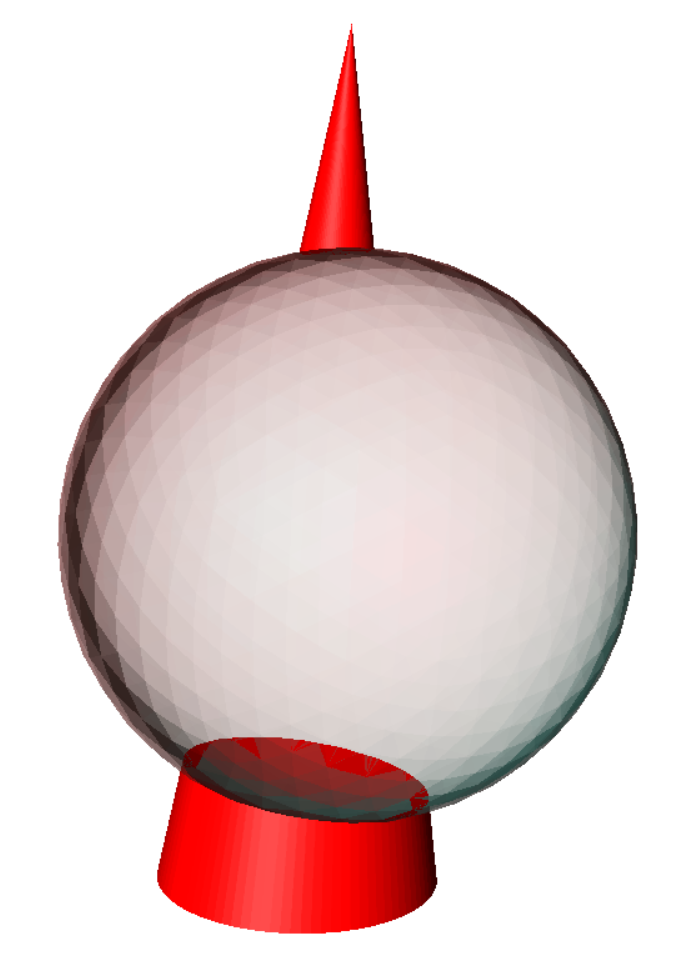


Рис. 33. Результат вычитания сферы из конуса

(прозрачная модель – исходная модель сферы, красная - результат операции)

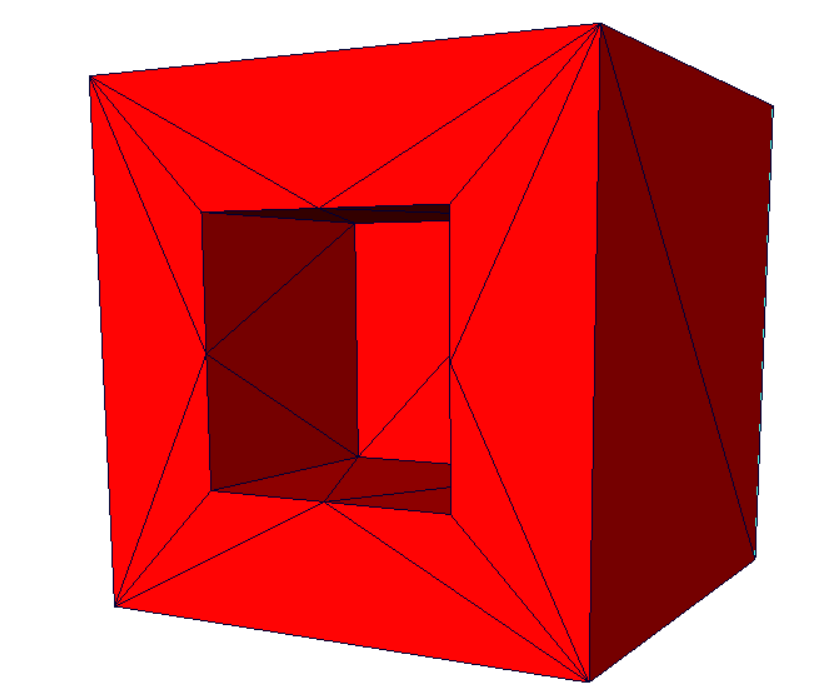


Рис. 34. Результат вычитания параллелепипеда из куба

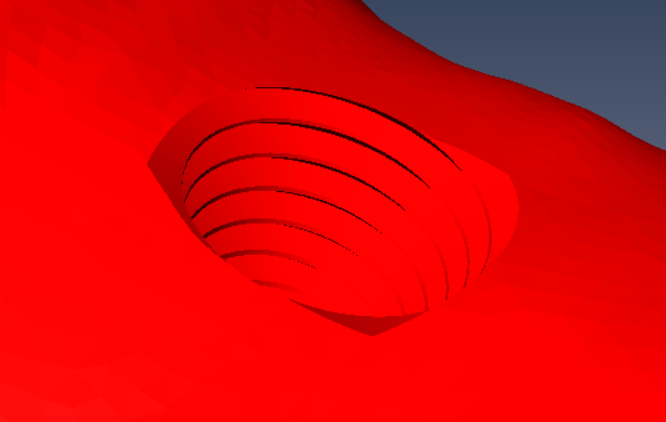
****

Рис. 35. Результат вычитания конического винта из кости позвоночника

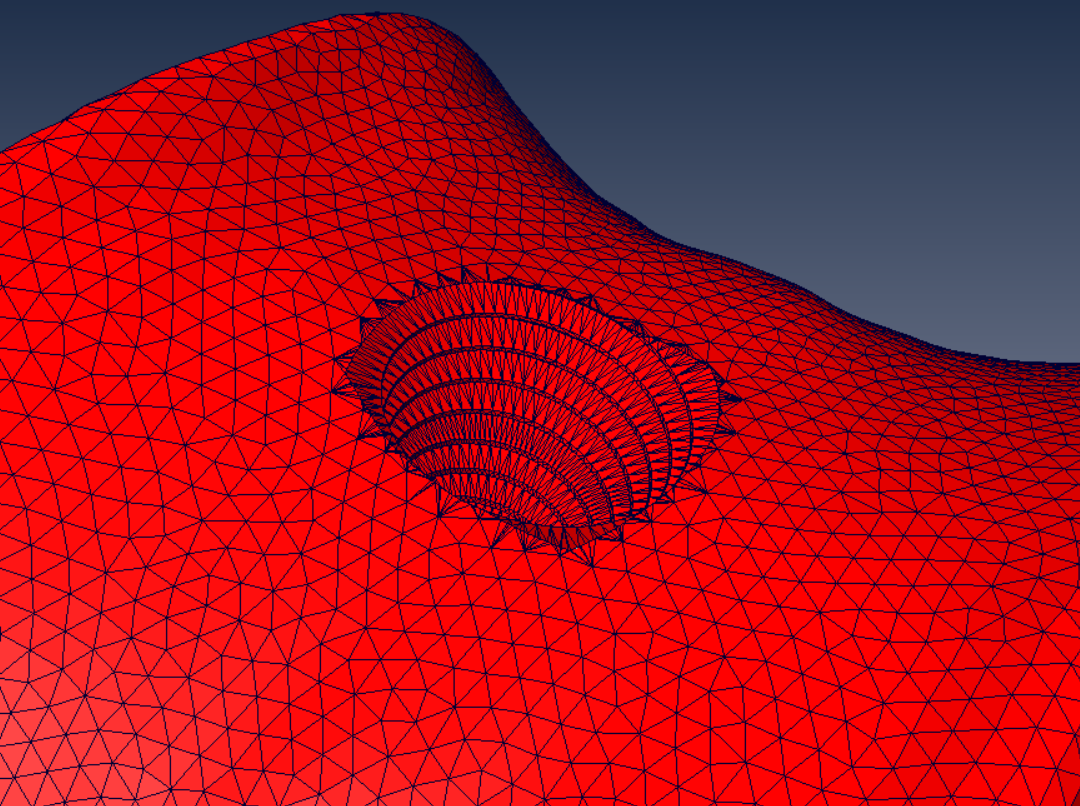
****

Рис. 36. Результат вычитания конического винта из кости позвоночника

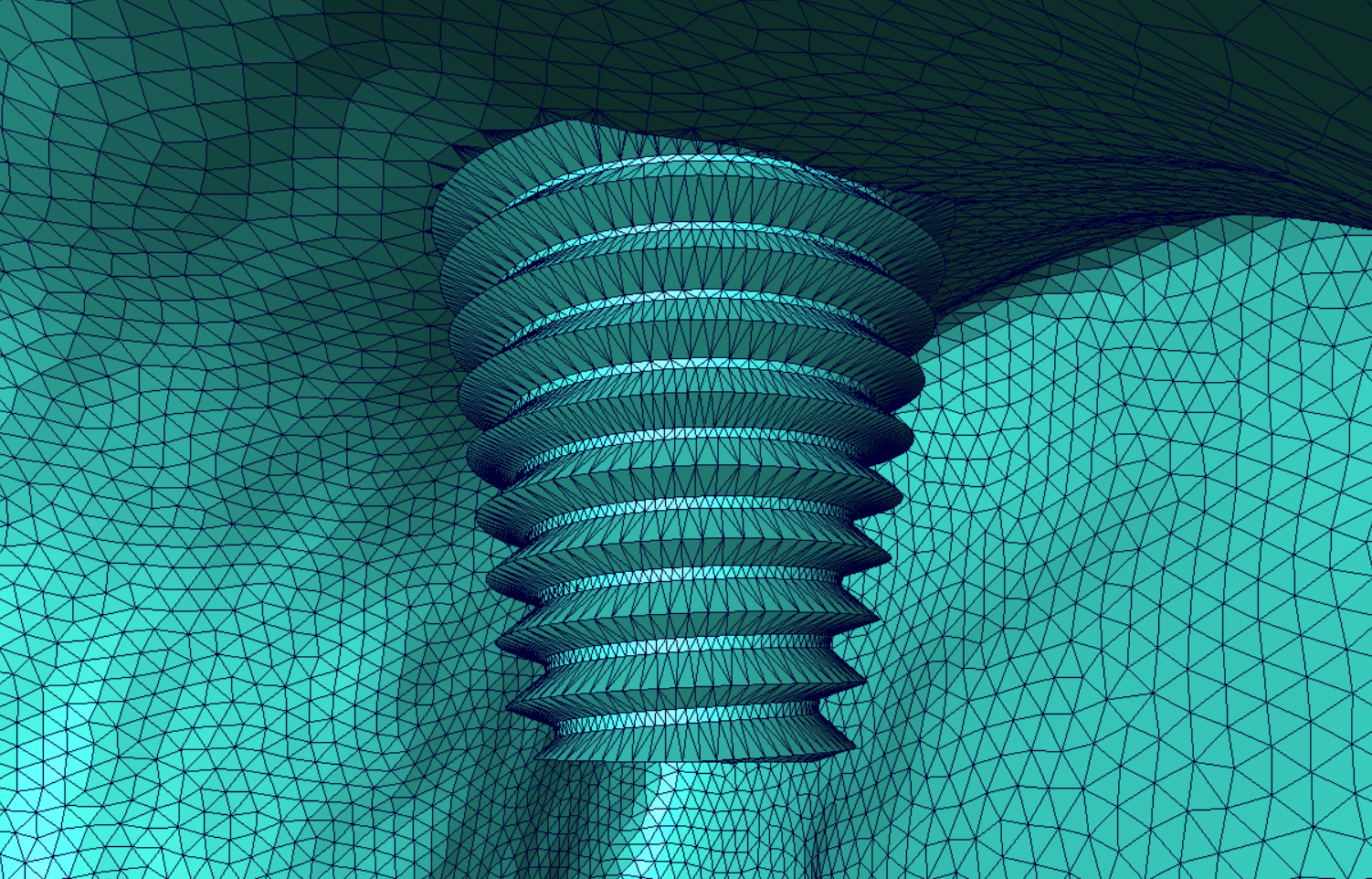


Рис. 37. Результат вычитания конического винта из кости позвоночника

(вид изнутри)

**4.5 Результаты тестирования**

Как видно из рисунков, все булевы операции со всеми тестовыми моделями были выполнены успешно. Анализ дефектов был проведён в программе Netfabb Basic [9]. Данная программа имеет инструментарий для работы с файлами в STL – формате и подготавливает их для 3Д – печати.

В моделях, которые являются результатами операций над кубом, параллелепипедом, а также конусом и сферой, не было обнаружено дыр. В результирующих моделях операций над костью позвоночника и коническим винтом было обнаружено три дыры, которые визуально не обнаруживаются. В программе Netfabb Basic в режиме автоматического ремонта все дыры были устранены, т. е. модель может быть распечатана на 3D – принтере.

Основным недостатком данной программы является скорость работы. Например, операция вычитания из кости винта заняла около пяти минут. Такое большое время работы связано с алгоритмом разбиения моделей на внешние и внутренние части, в котором происходит поиск пересечения треугольников одной модели со всеми треугольниками другой. В модели кости имеем 95 846 тысяч треугольников, винта – 9 300. Таким образом, программе надо выполнить 95 846 ∙ 9 300 = 890 000 000 операций пересечения треугольников. Устраняется это путём ввода структур данных, которые позволят сократить количество операций пересечения в сотни раз.

Результаты тестирования приведены в сводной таблице 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тестируемые модели  (количество треугольников) | Операция | Количество дыр | Non-manifold edges\* | Время операции, с |
| Конус/сфера  (378/1280) | Объединение | 0 | 0 | <1 |
| Пересечение | 0 | 0 | <1 |
| Вычитание | 0 | 0 | <1 |
| Куб/параллелепипед  (12/12) | Объединение | 0 | 0 | <1 |
| Пересечение | 0 | 0 | <1 |
| Вычитание | 0 | 0 | <1 |
| Кость позвоночника/  Конический винт  (95986/9294) | Объединение |  |  | 32 |
| Пересечение |  |  | 32 |
| Вычитание |  |  | 32 |

Таблица 1. Результаты тестирования

\*Non-manifold edges – это ребра, которые включены в более, чем два треугольника

**Заключение**

Написанная программа успешно выполнила булевы операции над всеми тестируемыми моделями. Все требования технического задания к программному продукту также были выполнены. Задача курсовой работы была достигнута.

В дальнейшем планируется развитие программного продукта в сторону его быстродействия. Для этого надо вводить структуры данных, изменять алгоритмы отдельных этапов программы.

Примечание: при написании программы не было использовано ни одной сторонней библиотеки, связанной с этой областью разработки, от других разработчиков, что повышает эффективность дальнейшей работы в плане понимания возможных ошибок и устранения недостатков программы.

**Литература**

[1] Майкл Ласло. Вычислительная геометрия и компьютерная графика на С++: Пер. с англ.­ – М.: «Издательство БИНОМ», 1997. – 304 с.: ил.

[2] Скворцов А.В. Триангуляция Делоне и ее применение. – Томск: Изд-во Том. ун-та, 2002. – 128 с.

[3] Ralf Hartmut Guting and Markus Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. FernUniversitat Hagen – Hagen – Germany, 1994.

[4] Kevin Weiler and Peter Atherton. Hidden Surface Removal Using Polygon Area Sorting. Cornell University – Ithac – New York, 1977.

[5] Netfabb Basic. - <https://www.netfabb.com>

[6] Совершенный код. Мастер-класс / Пер. с англ. — М. : Издательско-торговый дом «Русская Редакция» ; СПб.: Питер, 2005. — 896 стр.