Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования Московский государственный технический университет имени Н.Э. Баумана

Курсовой проект

по учебной дисциплине

«Численные методы»

На тему:

«Аппроксимация данных из специализированного медицинского формата»

Выполнил:
Студент группы ИУ9-72
_____ Белогуров А.А.
«____» _____ 2018 г.
Руководитель курсового проекта:
_____ Домрачева А.Б.
«____» _____ 2018 г.

Москва, 2018 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 3

1. Обзор предметной области. 5
   1.1. Трёхмерное моделирование. 5
   1.2. Триангуляция. 7
   1.3. Принципы конструктивной геометрии. 9

2. Формат описания поверхностной геометрии STL. 12
   2.1. Построение файла. 12
      2.1.1. ASCII STL. 13
      2.1.2. Binary STL 14

3. Проектирование интерфейса и его реализация. 15
   3.1. Выбор 3D библиотеки для визуализации трёхмерных моделей. 15
   3.2. Обнаружение столкновений объектов и их обработка. 16
   3.3. Применение конструктивной геометрии к объектам неправильной формы. 21

4. Тестирование. 24
   4.1. Конструктивная геометрия. 24
   4.2. Коллизии объектов. 25

ЗАКЛЮЧЕНИЕ 28

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 29

ВВЕДЕНИЕ

Современный мир информационных технологий трудно представить без использования трёхмерного моделирования, которое уже прочно вошло в многие сферы нашей жизни. С помощью него возможно визуализировать объекты, которые в дальнейшем применяются в кинематографе, транслируются по телевидению, прорабатываются в компьютерных играх, широко используются в медицине и различных областях классических наук.

Визуализация в медицине развивается в двух основных направлениях:

- точечная или комплексная томография;
- конструирование и создание протезов.

Современные приборы 3D-сканирования позволяют обнаружить дефекты органов и тканей, которые скрыты при простом рентгене или УЗИ. Появление таких технологий сделало возможным определение заболевания в тех ситуациях, когда ранее проводились диагностические операции. Широкое распространение они приобрели в стоматологии и челюстно-лицевой хирургии. Для удобства обращения с такими технологиями современные больницы не ограничиваются компьютерными макетами, а приобретают принтеры для объемной печати.

Воплощенный в жизнь результат томографии может стать основой для создания импланта, например, зуба, который будет идеально подходить по размерам пациенту. В более сложном варианте технология помогает смоделировать протез конечности, слуховой аппарат, модель вен и нервов, и даже искусственный сердечный клапан. Активно развивается биопечать – в ней вместо красок используются живые человеческие клетки. Но первый этап конструирования остается за компьютерными 3D программами. Здесь, как и при построении мультипликационных героев, используется полигональное моделирование. Искривление пластин показывает дефекты тканей. Воздействие на фрагменты позволит создать объемную фигуру идеального импланта, а

вращение и передвижение частей покажет, как будет двигаться протезированная рука.

Таким образом, можно сразу определить актуальность моей работы. Она заключается в том, что современный мир невозможно представить без трёхмерного моделирования. Его использование в медицине позволяет решать множество задач, упрощает визуализацию множества данных и является незаменимым инструментом при подготовке к операциям и другим различным исследованиям.

Объектом исследования являются трёхмерные модели протезов и их взаимодействие с моделями человеческого тела. Предметом исследования является отображение результатов на любом устройстве с помощью графического интерфейса.

В ходе работы должны быть выполнены следующие задачи:

- обзор предметной области и принципов конструктивной геометрии;
- исследование STL формата для построения трёхмерных моделей;
- проектирование интерфейса и его реализация;
- тестирование на различных входных данных.

Целью данной курсовой работы является разработка программных модулей для использования в медицинских целях, которые позволят визуализировать различное взаимодействие трёхмерных моделей.

1. Обзор предметной области.

1.1. Трёхмерное моделирование.

В предыдущем разделе было описано, как важно и актуально использование 3D-моделирования, но стоит более подробно разобраться в текущей предметной области и уделить внимания его деталям, которые в дальнейшем могут пригодиться для реализации программы.

3D-моделирование – это процесс создания трёхмерной модели объекта. Его задачей является разработка визуального объёмного образа желаемого объекта. При этом модель может как соответствовать объектам из реального мира (автомобили, здания, астероид), так и быть полностью абстрактной (проекция четырёхмерного фрактала).

Графическое изображение трёхмерных объектов отличается тем, что включает построение геометрической проекции трёхмерной модели *сцены* на плоскость (например, экран компьютера) с помощью специализированных программ.

Далее будут рассмотрены основные шаги для получения трёхмерного изображения на плоскости. **Моделирование** – создание трёхмерной математической модели сцены и объектов в ней. В настоящее время существует множество источников, которые предоставляют свободный доступ к различным моделям в разных форматах. Кроме этого, для получения 3D-моделей объектов из реального мира могут использоваться различные приборы для 3D-сканирования. Как правило, полученный объект строится с помощью множества полигонов, которые могут быть представлены в виде треугольников или четырёхугольников. Чем больше таких полигонов, тем точнее может быть представлена модель, однако не стоит забывать об увеличении потребляемой мощности для отображения и других вычислениях при увеличении такого рода данных. Поэтому оптимальным решением в таких случаях является соблюдение

границы, которая в рамках вычислительной задачи будет лучшим решением как для представляемой точности, так и для требуемой мощности.

Следующим шагом является **текстурирование** – назначение поверхностям моделей растровых или процедурных текстур (подразумевает также настройку свойств материалов – прозрачность, отражения, шероховатость и пр.). В большинстве случаев объектам придаётся такая текстура, которая соответствует им в реальном мире.

Далее определяется **освещение** – установка и настройка источников света, которых может быть несколько. Используется для более реалистичного поведения реализуемых моделей.

В зависимости от реализуемой задачи может использоваться **анимация** – придание движения объектам. Как правило, определяется цикл, в котором модель следует определенным алгоритмам в каждый момент времени.

Аналогично приведённой выше анимации в некоторых случаях может использоваться и **динамическая симуляция** – автоматический расчёт взаимодействия твёрдых/мягких тел и частиц с моделируемыми силами гравитации, ветра, выталкивания, а также друг с другом. В зависимости от используемых программ и математических пакетов она может быть уже реализована и подключена с помощью настраиваемых опций.

Одним из самых главных шагов моделирования является **рендеринг** или визуализация – построение проекции в соответствии с выбранной физической моделью. Именно здесь заканчиваются все вычисления на реализуемых входных данных и генерируется картинка, на которой будет виден результат всех ранее применённых шагов. Далее производится вывод полученного изображения на устройство вывода, которым может быть 3D-принтер, монитор компьютера или экран любого другого устройства.

Стоит отметить, что описанные выше шаги рекомендуется реализовывать в предложенном выше порядке. Однако, например, если в процессе рендеринга

потребуется изменить положение модели в пространстве или параметры его освещения, то будут выполнены следующие действия:

- рендеринг → изменение положения модели в пространстве → рендеринг;
- рендеринг → изменение параметров освещения → рендеринг.

Описанные выше шаги играют фундаментальную роль в построение математических моделей. Они могут быть реализованы вручную или с помощью готовых инструментов.

1.2. Триангуляция.

Выше было сказано, как объекты могут быть представлены в трёхмерном моделировании, а именно с помощью четырёхугольных и треугольных полигонов. Разберём более подробно способы построения таких поверхностей.

Триангуляцией называется планарный граф, все внутренние области которого является треугольниками. Задачей построения триангуляции по заданному набору точек называется задача соединения заданных точек непересекающимися точками так, чтобы образовалась триангуляция. Но задача построения таких треугольников является неоднозначной, поэтому необходимо выяснить, как именно должна строится триангуляция.

В настоящее время известно два наиболее популярных способа. Для первого из них необходимо, чтобы сумма длин всех рёбер была минимальна среди всех возможных триангуляций, построенных на том же наборе точек.

Вторым способом является построение триангуляции Делоне. Для этого необходимо, чтобы внутрь окружности, описанной вокруг любого построенного треугольника, не попадала ни одна из заданных точек триангуляции, и сама триангуляция была выпуклой. Благодаря выполнению этих свойств треугольники получаются по своей структуре и размеру похожими между собой, что позволяет построить более ровную и плавную поверхность в итоге.

Рассмотрим алгоритм построения триангуляции Делоне более подробно:

1. Происходит создание суперструктуры, которая будет покрывать все точки.
2. Выполняем цикл по n точкам для пунктов 3-5.
3. Очередная n-ая точка добавляется в триангуляцию следующим образом. Вначале производится локализация точки в зависимости от реализуемого алгоритма.
   3.1. Если точка попадает в эпсилон-окрестность любой другой вершины триангуляции, то она отбрасывается.
   3.2. Если точка попала на некоторое ребро, то оно разбивается на два новых, а оба смежных с ребром треугольника также делятся на два меньших. 5
   3.3. Если точка попала строго внутрь какого-нибудь треугольника, он разбивается на три новых.
   3.4. Если точка попала вне триангуляции, то строится один или более треугольников. (Так как на первом шаге была построена суперструктура, то все точки заведомо будут лежать внутри нее. Поэтому выполнение этого пункта невозможно)
4. Проводятся локальные проверки вновь полученных треугольников на соответствие условию Делоне и выполняются необходимые перестроения.

Самым быстрым алгоритмом построения триангуляции Делоне является итеративный алгоритм с динамический кэшированием поиска (Рисунок 1.2.1).

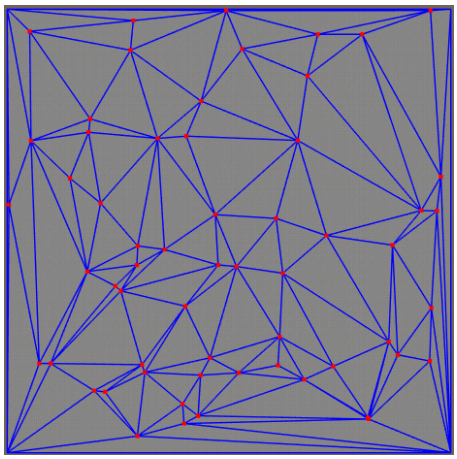

Рисунок 1.2.1 – Результат итеративного алгоритма триангуляции динамический кэшированием поиска

Его главная идея заключается в построение сетки динамического размера, с помощью которой будет происходит локализации новых точек триангуляции. Ячейки содержат ссылки на близлежащий треугольник, за счет чего уже построенный треугольник из кэша может быть получен за O(1).

1.3. Принципы конструктивной геометрии.

Конструктивная блочная геометрия или Constructive Solid Geometry(CSG) – технология, используемая в моделирование твёрдых тел, результат которой мы можем видеть каждый день в реальном мире, но не обращать на это внимания.

С помощью конструктивной геометрии возможно создать сложную поверхность или объект с помощью булевых операций для комбинирования нескольких объектов между собой (Рисунок 1.3.1).

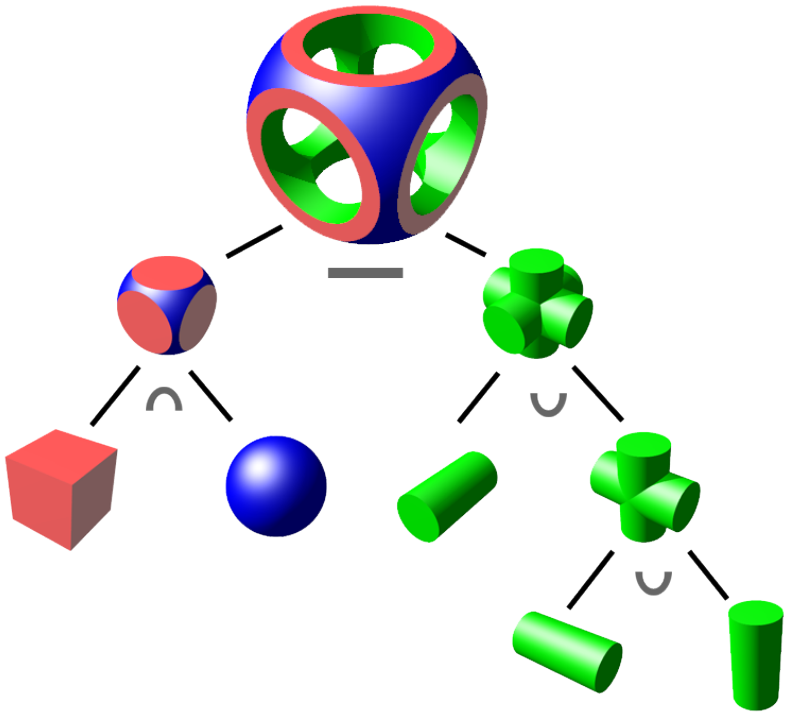

Рисунок 1.3.1 – Пример использования конструктивной геометрии

Она позволяет более просто математически описать сложные объекты, хотя не всегда операции проходят с использованием только простых тел. Так, часто с помощью конструктивной блочной геометрии представляют модели или поверхности, которые выглядят визуально сложными; на самом деле, они являются не более чем умно скомбинированными или декомбинированными простыми объектами.

Как правило, такие объекты называются **примитивами**. Они представляют собой тела с простыми формами: куб, цилиндр, призма, пирамида, сфера и другие. Набор доступных примитивов зависит от реализуемого программного пакета. Так, некоторые программы позволяют создание конструктивной блочной геометрии на основе кривых объектов, а некоторые нет.

Построение более сложного объекта происходит путём применения к описаниям объектов булевых (двоичных) операций на множествах — объединение, пересечение и разность.

Так, например, применение булевой разности к таким объектам, как куб и сфера, образует новый объект (Рисунок 1.3.2).

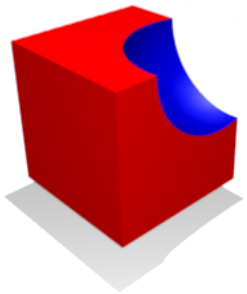

Рисунок 1.3.2 – Применение булевой разности к кубу и сфере

Использование конструктивной геометрии является одним из самых важных принципов в трёхмерном моделировании, и позволяет создавать множество сложных объектов, которые в дальнейшем могут использоваться в научных и медицинских целях.

2. Формат описания поверхностной геометрии STL.

2.1. Построение файла.

STL – формат файла, который используется для описания поверхностной геометрии и хранения трёхмерных моделей объектов. Разработан и представлен в 1987 году американскими разработчиками 3D systems. В настоящее время данный формат активно используется в медицинских целях и является одним из основных способов для печати моделей на 3D-принтерах. Кроме этого, 3D-печать может преобразовать реальный объект в STL-формат, который в дальнейшем можно будет дорабатывать и использовать в трёхмерном моделировании (Рисунок 2.1.1).

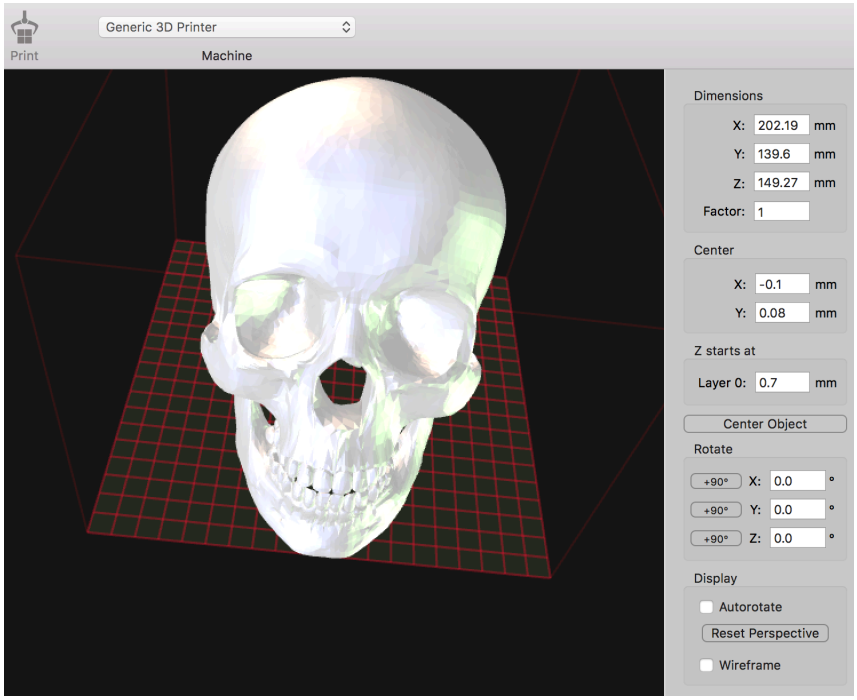

Рисунок 2.1.1 – Трёхмерная модель черепа в STL-формате [1]

Как видно выше на рисунке, STL-файл для формирования модели формирует список треугольных граней и их нормалей. Такой подход называется триангуляцией.

STL может быть текстовым (ASCII) или двоичным (Binary) файлом. Разберем более подробнее, как строится каждый из них.

2.1.1. ASCII STL.

Текстовый STL-файл формируется следующим образом (Листинг 2.1.1.1).

Листинг 2.1.1.1 – Пример текстового STL-файла

```
1.  solid name
2.  …
3.     facet normal nx ny nz
4.       outer loop
5.         vertex v1x v1y v1z
6.         vertex v2x v2y v2z
7.         vertex v3x v3y v3z
8.       endloop
9.     endfacet
10. …
11. endsolid
```

Первая строка всегда начинается с *“solid name”,* где *name* – имя представленной модели, может быть опущено, но всё равно должен стоять пробел после *solid.*

Далее файл продолжается произвольным числом треугольников, описываемых следующим образом (Листинг 2.1.1.1 строки 3-9), где каждое *n* и *v* – число с плавающей точкой, представленной в виде “-2.64800e-002”. Первая строчка *“facet normal nx ny nz”* описывает вектор нормали с тремя координатами к текущей грани. По правилам, такая нормаль должна быть единичным вектором, направленным от объекта. В большинстве программ она может быть установлена в (0, 0, 0), и программа автоматически рассчитает нормаль на основе порядка вершин треугольника, используя правило правой руки. Далее начинается описание одной треугольной грани, которая характеризуется тремя точками в пространстве.

Стоит отметить, что ASCII STL-файл имеет большой объём для описания сложных трёхмерных моделей, поэтому более распространён именно двоичный формат данного файла. Текстовые файлы используется в основном для описания различных примитивов.

2.1.2. Binary STL.

Двоичный STL-файл представляет собой последовательность байтов построенных определенным образом (Листинг 2.1.2.1).

Листинг 2.1.1.2 – Пример двоичного STL-файла

```
1. Name[80] – 1 byte
2. Number of triangles – 4 bytes
3. /* For each triangle */
4. Normal – real32 (4 bytes)
5. Vertex1 – real32 (4 bytes)
6. Vertex2 – real32 (4 bytes)
7. Vertex3 – real32 (4 bytes)
8. Attribute Byte – 2 bytes
```

Первая строка начинается с заголовка файла, ограниченного в 80 символов. Он так же может игнорироваться, но не должен начинаться с *“solid”,* так как с этой последовательности начинается ASCII STL файл. Далее идет 4 байтовое беззнаковое целое число, указывающее количество треугольных граней в данном файле. Далее аналогично текстовому формату описывается информация о каждой поверхности (Листинг 2.1.1.2 строки 4-8). Последняя строка для каждой грани характеризуется 2 байтами *“attribute byte”,* которая служит для описания цвета, но как правило, это значение обычно равно нулю, так как большинство программ не понимают других значений.

3. Проектирование интерфейса и его реализация.

3.1. Выбор 3D библиотеки для визуализации трёхмерных моделей.

Прежде всего стоит определиться, с помощью каких именно инструментов будет реализована данная работа. Чтобы это выяснить, необходимо более подробно проанализировать поставленную цель и определить те задачи, которые мы хотим решить.

Во-первых, это реализация сцены, где присутствуют следующие объекты: имплант любой примитивной формы, любая часть человеческого тела, внутри которого находится кость. Далее реализовать возможность прохождения импланта сквозь кожу, но определять момент соприкосновения с костью и останавливать движение.

Во-вторых, это применение конструктивной геометрии для объектов неправильной формы, загружаемых из STL-файлов.

Так как работа предполагает просмотр результата с помощью графического интерфейса, то далее будут рассматриваться только WEBGL библиотеки. В этой области на данный момент существует два основных движка: THREE.JS и BABYLON.JS. Из названий можно легко понять, что они реализованы на языке программирования JavaScript.

С помощью них можно создавать, изменять и оперировать с различными 3D-объектами. Обе библиотеки используют именно тот пошаговый алгоритм построения сцены, который упоминался в разделе **Трёхмерное моделирование**. Кроме прочего, данные библиотеки позволяют подключать различные модули, которые добавляют новые функции. В нашем случае, это будет работа с STL форматом и использование конструктивной геометрии.

Все используемые материалы являются программным обеспечением с открытым исходным кодом, что позволяет более подробно разобраться в обоих движках. Также они оба активно развиваются, имеют хорошую документацию,

большое сообщество разработчиков и набор различных примеров, что упрощает их изучение.

3.2. Обнаружение столкновений объектов и их обработка.

В технической литературе часто под термином “столкновение” используется термин “коллизия” (от англоязычного слова *collision*). Таким образом, обнаружение столкновений (collision detection) – это вычислительная проблема обнаружения пересечений между собой двух или больше объектов.

Прежде, чем перейти к решению этой проблемы, необходимо определить еще один термин. Полигональная сетка (*polygon mesh*) – это совокупность вершин, рёбер и граней, которые определяют форму многогранного объекта в трёхмерной компьютерной графике и объёмном моделировании (Рисунок 1.1.1). Далее для простоты обозначения будем обозначать такие объекты термином *mesh.*

Так как вся работа должна происходить с загружаемыми STL моделями (Листинг 3.2.1), то все они в последствии преобразуются в *mesh*, который можно отобразить на экране.

Листинг 3.2.1 – Загрузка STL модели куба

```
1. BABYLON.SceneLoader.ImportMesh("../stl_models/","Cube.stl", scene,
   function (newMeshes) {
2.     var mesh = newMeshes[0];
3.     var material = new BABYLON.StandardMaterial("myMaterial", scene);
4.
       myMaterial.diffuseColor = new BABYLON.Color3(0, 0.5, 0.3);
5.     mesh.material = myMaterial;
6.     objects.push(mesh);
7. });
```

Загрузка STL модели является асинхронным процессом для того, чтобы не блокировать другие действия, выполняемые со сценой. В теле метода *importMesh()* загружается новый mesh, к которому применяется материал с заданными параметрами.

Для решения проблемы со столкновениями будет использоваться библиотека BABYLON.JS, так как у неё по умолчанию реализована работа с обнаружением коллизий с помощью методов *mesh.intersectMesh()* и *mesh.intersectsPoint()*. Из названий понятно, что они позволяют определить факт столкновения двух различных *mesh* или *mesh* с точкой в пространстве.

Но существует большое различие в работе двух этих методов. Первый из них образует так называемый *bounding box* – это параллелепипед, в который вписан каждый из объектов (Рисунок 3.2.1). И далее уже учитывается пересечение этих параллелепипедов, что позволяет сократить вычислительные затраты. Для медицинских целей такой подход категорически не подходит, так как в этой области необходима очень большая точность.

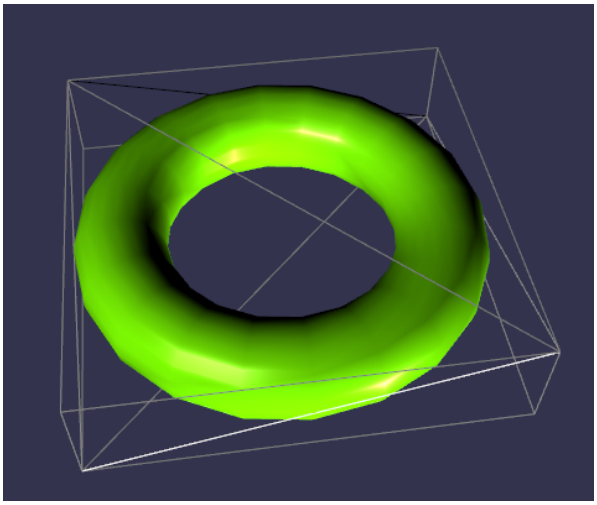

Рисунок 3.2.1 – Bounding box вокруг тора

Второй метод определяет столкновение объекта именно с произвольной точкой. Поэтому для решения проблемы определения коллизий был выбран следующий подход:

1. При загрузке трёхмерной модели из STL формата вычислять все вершины данной полигональной сетки и помещать их в массив.
2. В процессе движения другого объекта запускается цикл, который проверяет факт столкновения себя и точек из массива вершин загруженной модели.

3. Если столкновение обнаружено, прекращать цикл и останавливать движение.
4. Иначе продолжать выполнение цикла.

Реализация данного алгоритма приведена ниже (Листинг 3.2.2, Листинг 3.2.3).

Листинг 3.2.2 – Вычисление вершин загружаемой модели

```
var vertexData = BABYLON.VertexData.ExtractFromMesh(mesh);
var positions = vertexData.positions;
var numberOfPoints = positions.length / 3;
for (var i = 0; i < numberOfPoints; i++) {
    var p = new BABYLON.Vector3(positions[i * 3], positions[i * 3 + 1], positions[i * 3 + 2]);
    legMeshVertices.push(p);
}
```

Листинг 3.2.3 – Обнаружение столкновения объектов

```
if (!enableCollision) {

    var isIntersect = false;

    for (var i = 0; i < legMeshVertices.length; i++) {
        if (currentMesh.intersectsPoint(legMeshVertices[i])) {
            isIntersect = true;
            break;
        }
    }

    if (isIntersect) {
        currentMesh.material.diffuseColor = new BABYLON.Color3(1, 0, 0);
    } else {
        currentMesh.material.diffuseColor = new BABYLON.Color3(0, 0.5,0.3);
        currentMesh.position.addInPlace(diff);
        startingPoint = current;
    }
} else {
    currentMesh.position.addInPlace(diff);
    startingPoint = current;
}
```

Но так как описание объекта в STL формате состоит из вершин треугольных граней, то для двух смежный граней будут упоминаться две одинаковые точки. Следовательно, массив вершин надо формировать только из уникальных точек объекта (Листинг 3.2.4).

Листинг 3.2.4 – Метод для удаления повторяющихся точек из массива

```
function uniqBy(verticies, key) {
    var index = [];
    return verticies.filter(function (item) {
        var k = key(item);
        return index.indexOf(k) >= 0 ? false : index.push(k);
    });
}
```

Таким образом, размер массива можно уменьшить более чем в четыре раза и увеличить быстродействие программы.

Результат операции нахождения пересечений можно видеть ниже (Рисунок 3.2.2 и Рисунок 3.2.3). В качестве объектов были использованы STL модели куба и кости. При определении факта пересечения передвигаемый объект меняет цвет на красный.

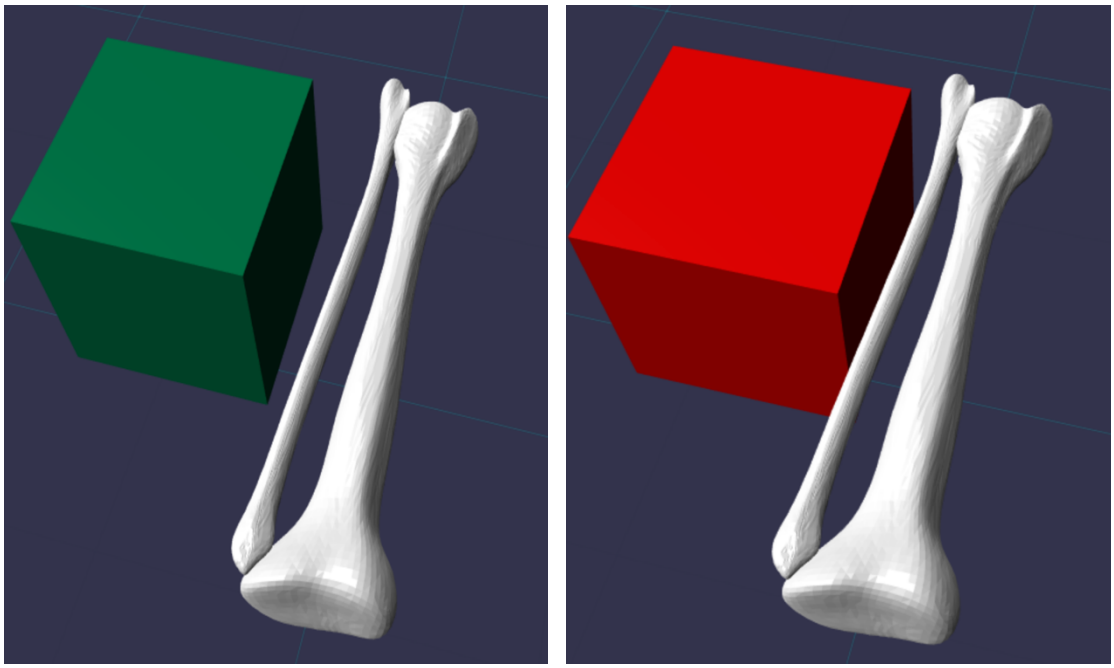

Рисунок 3.2.2 – Определение пересечения двух объектов

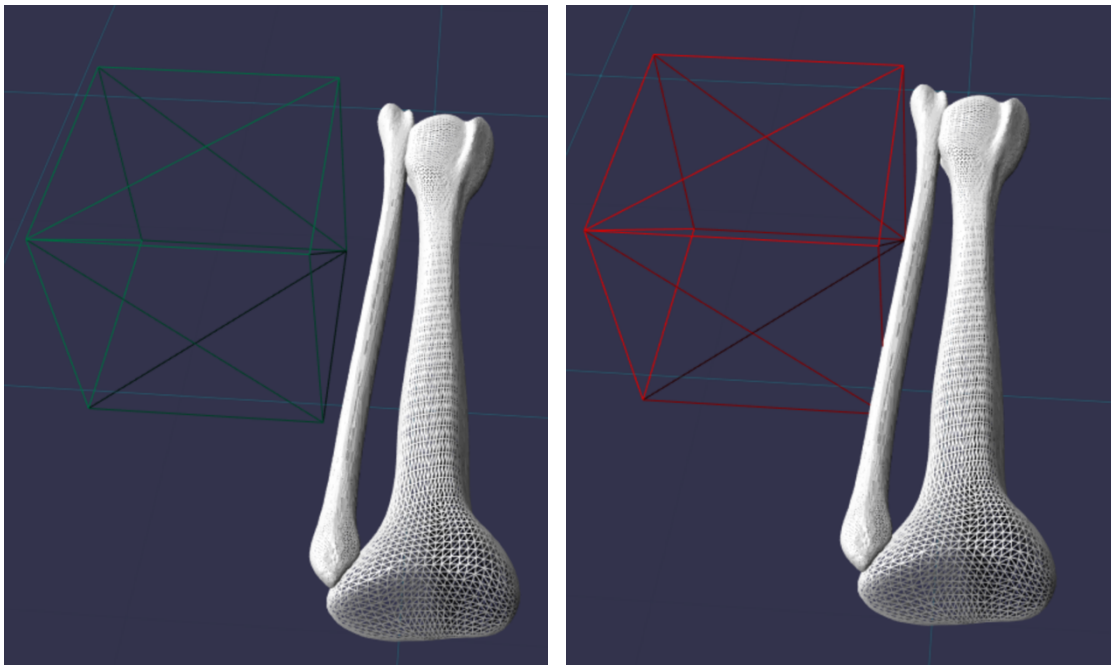

Рисунок 3.2.3 – Определение пересечения двух объектов для триангулированных поверхностей

Далее с помощью интернета была найдена модель ноги, внутрь которой будет помещена кость. Таким образом, решение проблемы, обозначенной в начале текущего раздела можно наглядно видеть на Рисунке 3.2.4.

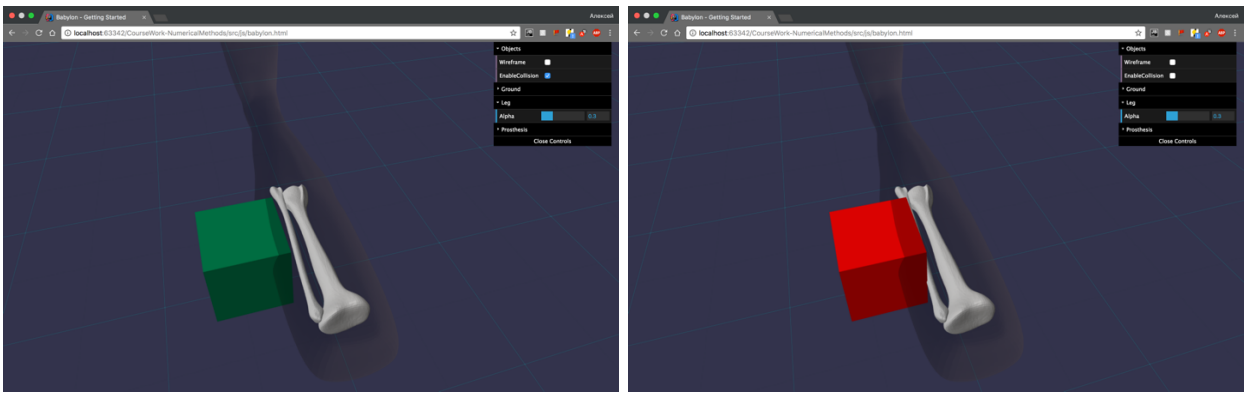

Рисунок 3.2.4 – Определение пересечения объекта с костью внутри ноги

Как можно заметить на рисунке выше, модель ноги изображена прозрачной, что позволяет более наглядно продемонстрировать факт

пересечения объектов. Значение прозрачности можно регулировать с помощью небольшого окошка, которое расположено в верхнем правом углу (Рисунок 3.2.4). Кроме этого, можно включать и выключать факт пересечения объектов, отображать полигональную сетку всех трёхмерных моделей на сцене и регулировать поворот протеза, в данном случае кубика, вокруг всех 3 осей координат.

3.3. Применение конструктивной геометрии к объектам неправильной формы.

Для решения этой задачи будет использована библиотека THREE.JS, так как для неё существует модуль, которая реализует всю работу конструктивной геометрии для *mesh* произвольной формы. Сам процесс при этом выглядит следующим образом:

1. Загрузить модели объектов из STL формата и получить на выходе их mesh.
2. Поместить mesh в положение, из которого будет выполняться одна из операций конструктивной геометрии.
3. Применить операцию вычитания/объединения/пересечения.
4. В зависимости от задачи, полученный mesh можно экспортировать в STL формат.

Реализация шага 3 представлена в Листинге 3.3.1:

```
1.  var mesh2 = meshes.pop(),
2.      mesh1 = meshes.pop();
3.  meshesGroup.remove(mesh1, mesh2);
4.  var meshCSG1 = THREE.CSG.fromMesh(mesh1),
5.      meshCSG2 = THREE.CSG.fromMesh(mesh2),
6.      result;
7.
8.      switch (operation) {
9.          case '1':
10.         default:
11.             result = meshCSG1.union(meshCSG2);
12.             break;
13.         case '2':
14.             result = meshCSG1.intersect(meshCSG2);
```

```
15.             break;
16.         case '3':
17.             result = meshCSG1.subtract(meshCSG2);
18.             break;
19.     }
20.     var material = new THREE.MeshPhongMaterial({color: 0xFAFAFA, specular:
21. 0x444444, shininess: 200});
22.     resultMesh = THREE.CSG.toMesh(result, material);
23.     meshes.push(resultMesh);
24.
25.     meshesGroup.add(resultMesh);
26. }
```

Сначала из массива объектов, извлекаются для последних mesh, которые в строчках 4-5 преобразуются в формат для конструктивной геометрии. Далее, исходя из номера операции, образуется новый объект, который преобразуются обратно в mesh и отображаются на экране (Листинг 3.3.1 строки 22-25).

Результат операции вычитания одного объекта из другого представлен на рисунке ниже (Рисунок 3.3.1).

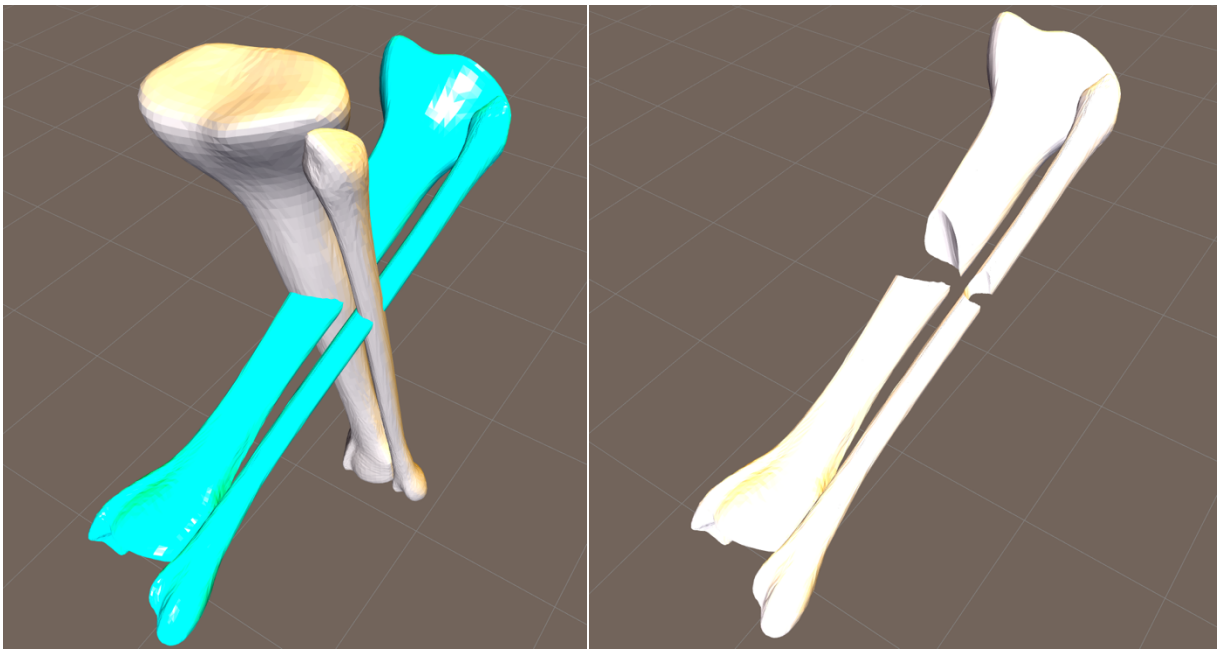

Рисунок 3.1.1 – Операция вычитания для двух объектов неправильной формы

Таким образом, с помощью конструктивной геометрии можно получать различные композиции отображаемых объектов, как правильной, так и

неправильной формы. Полученный результат можно успешно использовать в других программах или в 3D-печати.

4. Тестирование.

4.1. Конструктивная геометрия.

Проведём тестирование операций конструктивной геометрии для различных тел неправильной формы и сравним полученный результаты с ожидаемыми. Все объекты загружаются из STL формата, которые свободно распространяются в интернете.

В качестве объектов были взяты модели ноги и кости. Для усложнения задачи один объект помещается внутрь другого, их исходное положение продемонстрировано на Рисунке 4.1.1.

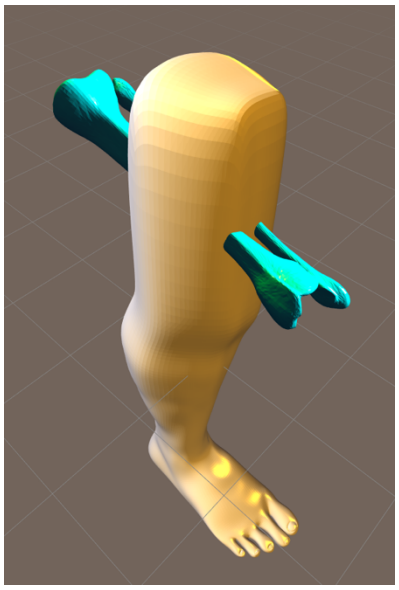

Рисунок 4.1.1 – Исходное положение объектов

Далее поочередно применим к ним операции конструктивной геометрии (Рисунок 4.1.2).

Рисунок 4.1.2 – Применение операций пересечения, вычитания и объединения

Как видно из рисунков выше, все операции конструктивной геометрии прошли успешно и удовлетворяют ожидаемым результатам. Таким образом, программное приложение для выполнения операций конструктивной геометрии можно использовать для решений многих задач, возникающих в медицине.

4.2. Коллизии объектов.

Далее необходимо протестировать возможность прохождения импланта через предметы различной формы. В качестве предметов будут использоваться различные STL модели костей человеческого тела, которые находятся в свободном доступе в интернете. Благодаря им можно максимально приблизить условия тестовой среды к реальным, отследить какие-либо дефекты и устранить их.

Ниже будут приведены примеры взаимодействия импланта с STL моделями костей разной формы.

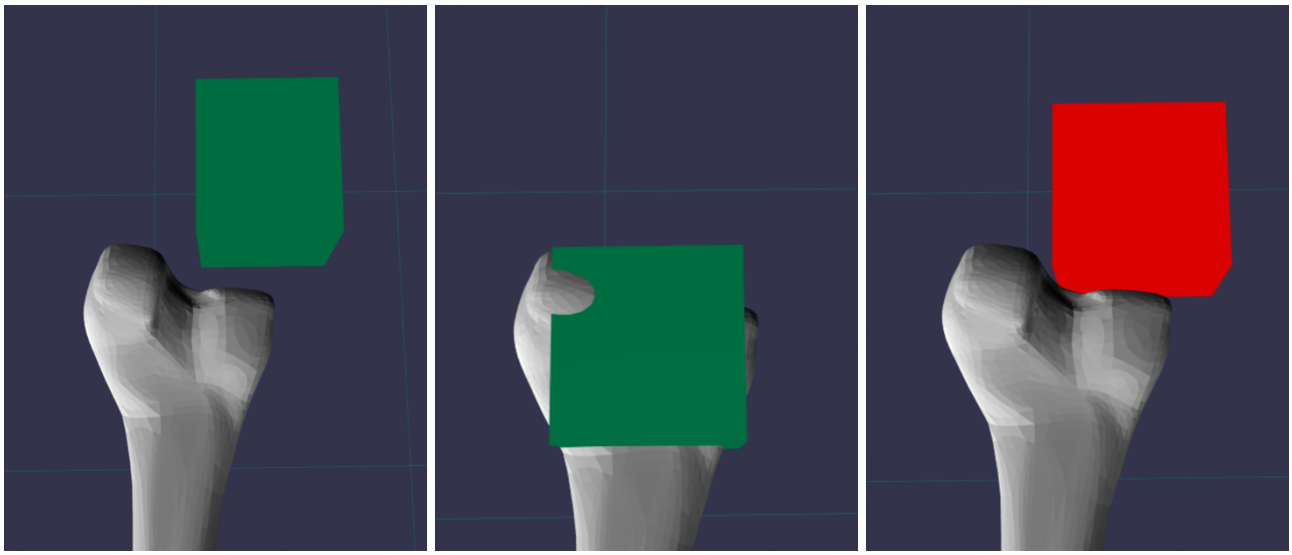

Рисунок 4.2.1 – Прохождение импланта сквозь STL модель №1

На Рисунке 4.2.2 будут изображены триангулированные поверхности для того, чтобы удостовериться, что объекты не проходят сквозь друг друга в случае отключения режима коллизии и все работает корректно.

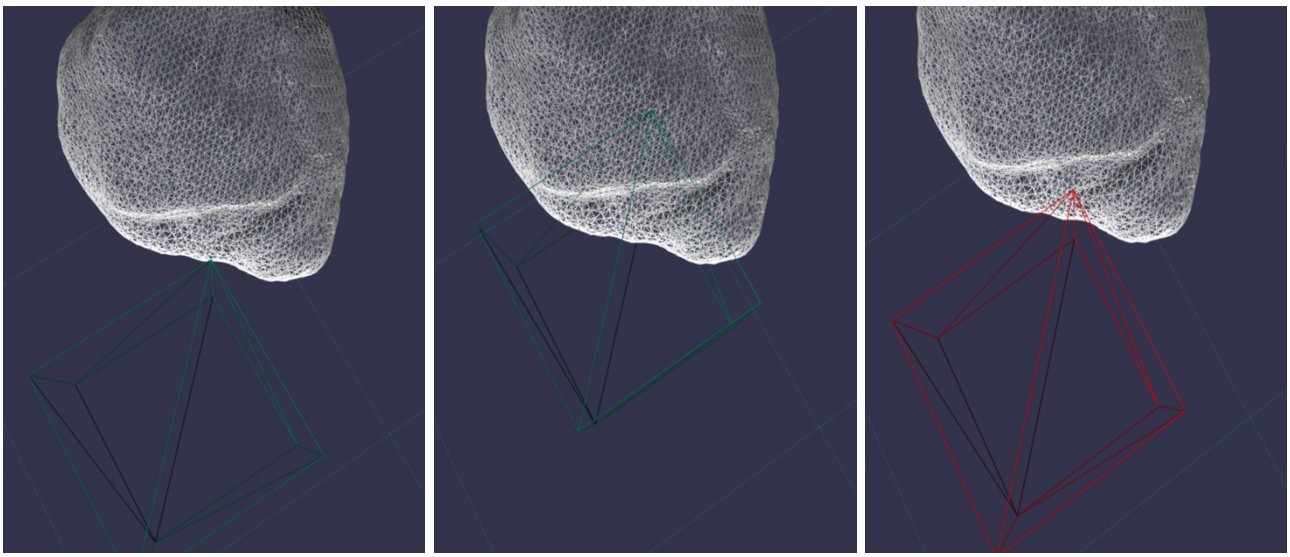

Рисунок 4.2.2 – Прохождение импланта сквозь STL модель №2

Аналогично предыдущему пункту, полученные объекты удовлетворяют ожидаемым результатам. В качестве импланта могут использоваться не только примитивы, но и различные трёхмерные модели сложной формы. Но также стоит

учесть тот факт, что чем более точнее представлены используемые объекты, тем правильнее может быть результат их взаимодействия.

ЗАКЛЮЧЕНИЕ

В ходе написания курсового проекта были выполнены следующие пункты:

- были изучены этапы построения трёхмерных сцен и моделей для их отображения на экране различных устройств, а также их представления в STL-формате;
- были исследованы основные принципы конструктивной геометрии;
- реализованы два программных модуля, один их которых демонстрирует работу с коллизией двух разных объектов, а второй отображает результат конструктивной геометрии для объектов неправильной формы;
- так как, вся работа проводилась исключительно с уклоном в медицину, то было проведено тщательное тестирование на реальных STL моделях.

Кроме прочего, для реализации программ были использованы библиотеки THREE.JS и BABYLON.JS, которые имеют огромный функционал для дальнейшего развития, добавления новых инструментов, использования уже готовых или их доработки для добавлений дополнительных сценариев использования.

Все полученные результаты были успешно протестированы и могут под наблюдением специалистов постепенно внедряться в уже существующие программные комплексы или участвовать в создании новых. Кроме медицинской сферы, полученные программы могут помогать решать различные задачи в таких областях, как строительство, обучение, конструирование различных передвижных комплексов и другие.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. STL File Format for 3D Printing. Available at: https://all3dp.com/what-is-stl-file-format-extension-3d-printing/, accessed 20.02.2018.
2. А. В. Скворцов. Триангуляция Делоне и её применение. Издательство Томского университета. 2002. 128 с.
3. Popular models – GrabCad – CadLibrary. Available at: https://grabcad.com/library, accessed 20.02.2018.
4. Biomechanics 2. Available at: http://biomechanika.fme.vutbr.cz, accessed 20.02.2018.
5. Официальная документация библиотеки THREE.JS. Available at: https://threejs.org/docs/, accessed 20.02.2018.
6. Официальная документация библиотеки BABYLON.JS. Available at: https://doc.babylonjs.com, accessed 20.02.2018.