Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования
**«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»**

Институт математики и компьютерных наук

Кафедра прикладной математики

Система удаленной визуализации трехмерных объектов с использованием сред виртуальной реальности. Описание модели

«Допущен к защите» Зав.кафедрой Логинов М.И.	Квалификационная работа на соискание степени магистра наук по направлению 02.04.01 Компьютерные науки Магистерская программа «Математическая кибернетика» студента гр. МКМ-240602 Левчука Георгия Ивановича
_____ «___»_____2016 г.	Научный руководитель Авербух Владимир Лазаревич, доцент, кандидат физико-математических наук

Екатеринбург
2016

РЕФЕРАТ

Левчук Г.И.
Система удаленной визуализации трехмерных объектов с использованием сред виртуальной реальности. Описание модели.
Дипломная работа, стр. 48, рис. 11, прил. 2, лит-ра 4

Ключевые слова: THREE.JS, WEBGL, JAVASCRIPT, HTML, КОМПЬЮТЕРНАЯ ВИЗУАЛИЗАЦИЯ, 3D МОДЕЛИРОВАНИЕ, 3D-ПРИНТЕР, ПОЛИГОН, ВИРТУАЛЬНАЯ РЕАЛЬНОСТЬ

Цель работы заключалась в реализации web-приложения с возможностью формирования 3D объектов и последующей работы с ними в качестве модели для 3D-принтера, а так же для дальнейшего использования в обучении студентов медицинских курсов и изучении других информационных объектов.

Web-приложение представляет собой программу в формате html с подключаемыми библиотеками Three.js и WebGL для 3D моделирования, с помощью которых происходит визуализация объектов, рендеринг компьютерной графики и формирование возможности просмотра 3D объектов в виртуальной реальности.

Вывод: В результате выполненной работы было реализовано приложение обеспечивающее формирование медицинских 3D моделей, а так же проводилось исследование некоторых свойств библиотек Three.js и WebGL.

Содержание:

Введение.

Программирование предоставляет безграничные возможности людям. Исследования, которые связаны с программирование позволяет уменьшить количество временных и физических затрат людьми. Таким образом, появляется множество разных приложений, сайтов и программ, которые позволяют пользователю решать определенные задачи: просмотр кино, решение дифференциальных уравнений, поиск нужной информации и многое другое. Говоря в целом, человеко-компьютерное взаимодействие позволяет увеличить качество жизни.

Представьте себе такую ситуацию - человек получает травму, такую, что больше он не может использовать часть своего тела из-за сложного перелома, например, перелом ноги или руки. Благодаря рентгену больного и другим анализирующим здоровье человека процедурам можно получить всю информацию о травмированной части скелета. На основе этой информации создаем 3D модель кости. Это представляет собой процесс формирования сложной 3D модели на нашем сайте. Сформированную 3D модель можно вращать, корректировать, если есть какие-то особенности кости (вмятины, прогибы и т.д.). Модель сложная, потому что кость не является какой-нибудь типичной геометрической фигурой как треугольник, куб и т.д.

Сформированную и отвечающую требованиям и особенностям строения скелета, 3D модель представляет собой файл, который можно передать на 3D-принтер и получить кость уже в осязаемом состоянии. Для пояснения скажу, что образованная трехмерная модель, является копией поврежденной кости до ее травмы.

Приложение по формированию 3D моделей костей позволят увеличить возможности работы медицинских специалистов, а так же повысить качество жизни людей. Я решил попробовать написать такой проект, чтобы реализовать, описанные выше функции приложения, а так же увеличить свои знания в области визуализации.

Было решено реализовать проект на основе языка программирования и библиотеки для работы с 3D графикой:

- WebGL;

- Three.js.

Эти инструменты были выбраны по причине повышения скорости формирования и отображения 3D моделей, потому что используют в работе видеокарту, а не операционную систему. А так же они позволяют запускать приложения во всех браузерах, благодаря возможности этих технологий работать на кроссбраузерном уровне.

Пользователи сайта будут действовать согласно определенному порядку действий:

- пользователь выбирает вид кости;

- предоставляется информация о специфичности строения кости травмированного человека;

- по выбору пользователем кости и данных о ее строении формируется 3D модель внутри трехмерной сцены.

После визуализации объекта, специалисту предоставляется возможность погружения в виртуальную реальность с помощью устройства – очков виртуальной реальности.В данный момент, разработано большое количество различных программ для очков виртуальной реальности, которые позволяют видеть руки или какие-то предметы с датчиками через эти очки. С помощью этих программ и устройств, у специалистов появляется возможность проверки правильности сформированной кости с травмированной костью до ее повреждения. Таким образом, появляется возможность проверять правильность сформированной кости.

Моя задача разработать такое web-приложение, у которого будет возможность моделирования различных 3D моделей костей для дальнейшей работы специалистов с этими объектами (3D объекты можно вращать, двигать, а так же углубляться в нее и просматривать внутреннее строение). Перед началом работы моего приложения на глобальном уровне, его будут тестировать медицинские специалисты. Если все тесты будут успешны, то будет одобрено использование данного приложения для формирования трехмерных моделей костей и дальнейшего вживления этих моделей людям.

Так же у сайта будет возможность просматривать кости и получать определенную информацию о них. Эта возможность в приложении будет повышать уровень знаний людей, посещающих медицинские курсы.

Постановка задачи.

Реализовать проект, который поможет инвалидам, людям, которые получили серьезные травмы костей во время боевых действий или других ситуаций. Первое, что могут ожидать люди после травм костей – это факт того, что они стали инвалидами. Но наше приложение дает возможность помочь врачам вылечить людей, у которых есть подобные травмы.

Было решено сформировать приложение, которое будет позволять визуализировать любую кость, и делать точной копией для замены травмированной.

Для реализации проекта были сформированы определенные задачи, которые определяют его функционал:

1. Реализация приложения с возможностью удаленного формирования трехмерной сцены;

2. Подключение 3D моделей из готовых библиотек в сцену, для дальнейших исследовательских работ;

3. Возможность просмотра трехмерных объектов в виртуальной реальности (данная задача решена Гвоздаревым Ильей Леонидовичем и отображена в его магистерской работе, которая имеет название «Система удаленной визуализации трехмерных объектов с использованием сред виртуальной реальности. Реализация взаимодействия с виртуальными объектами»).

Для реализации этих целей появилась потребность выбора удобных в работе с 3D графикой библиотек, и в связи с использованием этих технологий появилась возможность удаленного подключения к приложению для работы с трехмерными пространствами и объектами. После работы с некоторыми

библиотеками, было принято решение остановиться на определенном выборе современного программного обеспечения:

- Three.js;

- WebGL;

- JavaScript.

План разработки приложения.

План – это последовательность действий, выполнение которой приведет к поставленной цели.

С помощью грамотного плана работы, можно сделать процесс исследования и разработки приложения более успешной для исследователя и получить полезный продукт для специалистов, которые будут работать с данным приложением.

Согласно поставленным задачам был сформирован план действий для реализации web-приложения:

- разработать удобный интерфейс приложения и включить в него сцену для работы с 3D объектами в трехмерном пространстве;

- реализовать процедуру подключения 3D моделей с дальнейшей работой над ними.

Сейчас я попытался описать общую картину того, что мы сформируем для решения нашей задачи. В дальнейшем будут рассказаны цели, ради которых мы решили заняться этой проблемой. Так же опишем инструменты, которые использовались в создания нашего приложения.

Цель разработки приложения.

Целью работы является улучшение уровня жизни людей, благодаря современным технологиям программного обеспечения и возможностям аппаратных средств. Так же это исследование может повлиять на развитие виртуального компьютерного мира.

На первый взгляд может показаться, что нет смысла использовать данное приложение людьми, но это не так. Сейчас попробуем объяснить, что это правильные мысли, но скажу, что web-приложение может увеличить уровень работы врачей и качество жизни людей.

Когда человек берется за определенную задачу, то он преследует важную цель. В данном случае, цель работы заключается в улучшении жизни пользователей, которые получили серьезную костную травму, будь то сильное раздробление какой-то кости или другой серьезный перелом.

Для полного понимания актуальности данной задачи приводится пример стандартной ситуации, в которой приложение может принести пользу:

– человек попал в аварию и повредил ноги, и узнает о том, что больше никогда не сможет ходить.

Но вскоре ему сообщают, что есть новая технология, которая позволит ему вновь чувствовать себя полноценным. Технология дорогая и сложная, и как это обычно бывает, есть риск, что произойдет что-то непоправимое. В таком случае выбор стоит перед самим человеком, и только он решает делать что-то или оставить все как есть. Сможет ли он пережить эту утрату или выберет вариант рискнуть и попробовать преодолеть себя и сделать невозможное. И если выбор человека пал на попытку рискнуть и все исправить, то ему надо пройти определенную процедуру подготовки перед сложной операцией.

Сначала нам нужна информация о скелете человека, то есть данные о кости: рентген, результаты осмотра врачей и другим источникам информации, которые помогут сформировать трехмерную модель поврежденной кости до ее травмы. Не обязательно уметь формировать кость внутри приложения, а вот главной задачей является подключение готовых трехмерных объектов и работа с ними, просмотра структуры строения кости и возможность изменения объекта, если он не совпал с костью, до ее повреждения. Об этом расскажем чуть-чуть подробнее.

Созданное приложение позволяет формировать кость для человека. Именно их мы сможем формировать в любых размерах и возможностью растягивать отдельные ее фрагменты, с возможностями просматривать кость в виртуальной реальности и погружаться во внутреннюю часть костей за счет одноименных очков.

Чтобы была возможность проверки правильной работы всего приложения, нужно смотреть эту кость в виртуальной реальности. То есть мы беем кость, берем очки виртуальной реальности и специальное устройство позволяющее видеть кость в виртуальной реальности или простого просмотра структуры костей. Таким образом, мы можем сравнивать 3D модель кости с точной копией поломанной кости, до ее повреждения. Если после проверки сформированная трехмерная модель не совпало с нормальной костью, то модель должна быть переделана. Если 3D объект полностью совпал, специалисты согласны проводить операцию и сам пользователь согласен в ней участвовать, то дальше происходит процесс формирования кости по ее трехмерной модели в 3D-принтере.

Формирование кости 3D-принтером по файлу с информацией о 3D модели будет происходить с помощью специального материала. Так называемый «специальный материал», из которого формируется новая кость, является легким металлом, специально чтобы была возможность вживлять его в тело

человека, и он будет нормально взаимодействовать с ним, при этом, никак не нанося травм людям.

Если все удастся, то данное приложение сможет помочь людям в еще одном очень важном вопросе. В положительном варианте это все может свести к тому, что у людей появиться возможность формировать не только кости, а любые человеческие органы и многое другое.

В противном случае у людей появиться возможность формировать любую 3D модель по своему усмотрению и каким-то образом этим пользоваться в дальнейшем, потому что возможности использования трехмерных моделей в компьютерном и реальном мире безграничны.

Разрабатываемое приложение будет позволять людям получать информацию по строению человеческого скелета. И в дальнейшем появляется возможность сформировать 3D модель кости любого существа, и в дальнейшем использовать эти модели для получения информации о них в операциях, проводимых с этим приложением, либо в изучении специальных медицинских курсов.

Чтобы все представляли общую картину, которая описывает идею написания нашего сайта, мы попытались сформулировать основные задачи, описать коротко общие моменты и подробнее рассмотреть сложные аспекты проблемы и ее решения. В дальнейшем будет подробно рассказано о том, как реализовывалось приложение с возможностью просмотра в виртуальной реальности и формирования 3D модели кости с примером (рис.1).

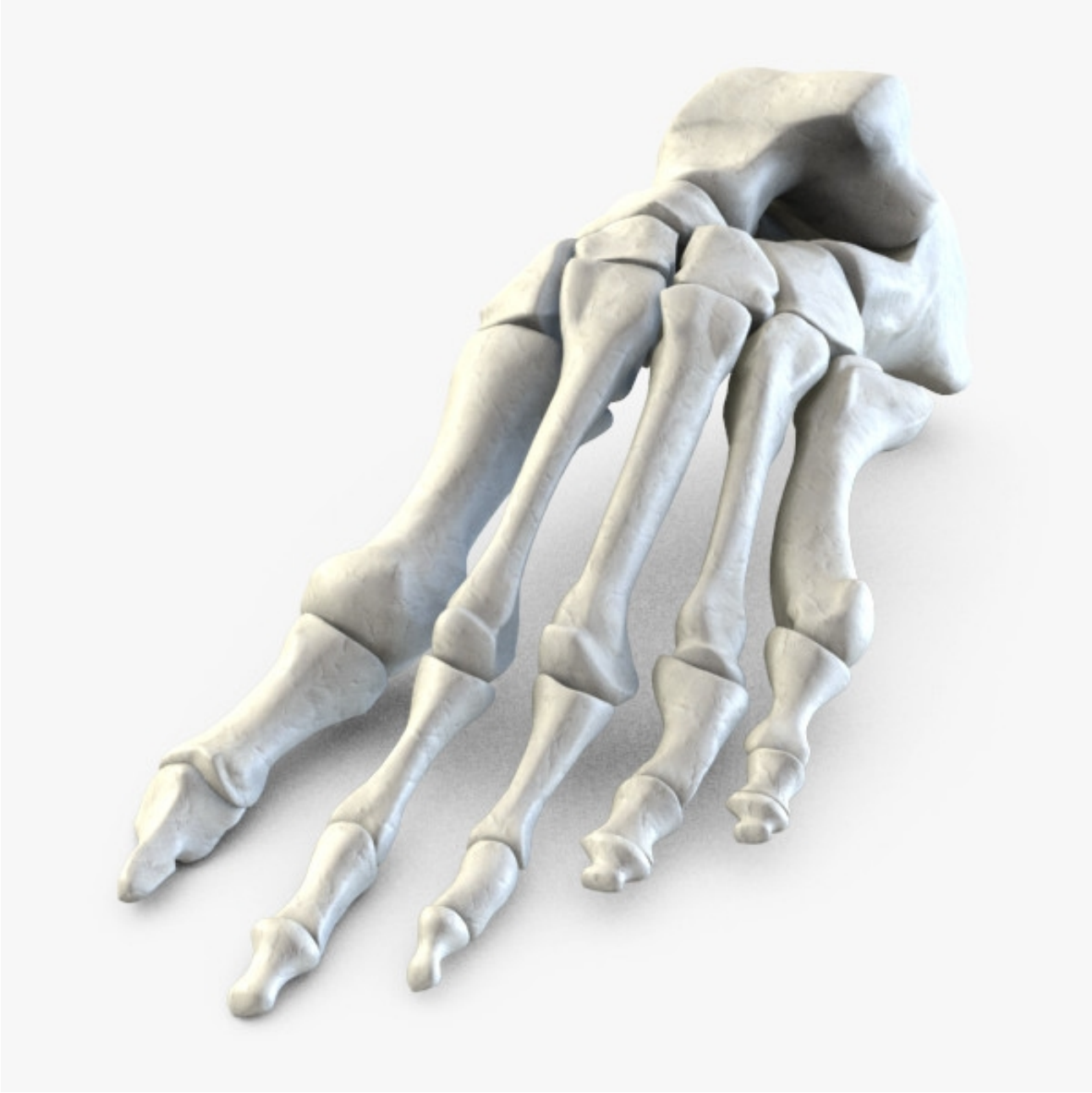

Рис1. Пример трехмерной модели кости.

Разработка приложения.

Для описания решения данной задачи я представил процесс подготовки технологии к ее реализации в определенном плане, который состоит из нескольких пунктов. Все фрагменты кода представлены в приложениях, отсылки на которые будут появляться в дальнейшем. И начнем с краткого описания техник, которыми пользовались для реализации приложения, а именно html-файла.

Глава 1. Формирование сайта

Проект создавался на базе двух систем для формирования сайта: html и JavaScript(JS).

Html - стандартизированный язык разметки документов во всемирной паутине. Большинство web-приложений содержит описание разметки на языке HTML. HTML-теговый язык разметки документов. Любой документ на этом языке представляет собой набор элементов, причем начало и конец каждого элемента обозначается специальными пометками тегами (тег - элемент языка разметки гипертекста). Текст содержащийся между тегами, отображается и размещается согласно свойствам указанным в начальном теге.. Так же есть пустые теги, самозакрывающиеся теги (
) спецсимволы(&имя) и другие сущности для возможности формирования новых возможностей сайта.

JavaScript – прототипно – ориентированный сценарный язык программирования. JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широко применяется в браузерах как язык сценариев для придания интерактивности веб-страницам.

Сильно глубоко, как и обещал, вдаваться в подробности не буду, а весь код приложения добавлен в «Приложение 1» .

Глава 2. Создание 3D модели

В этой главе будет описан процесс создания нашей 3D модели. Есть несколько способов формирования 3D объектов для дальнейшей работы с ними: «360 градусов», 3D модель и формирование 3D модели внутри сайта.

A) Первый способ достаточно прост, как и в понимании того, как получить нашу модель, так и в самой реализации этого способа. Тем не менее, есть и свои сложности этого способа, в отличие от других методов реализации 3D объектов. В чем заключается идея формирования 3D модели методом, который чаще всего называют «360 градусов».

Идея состоит в том, что фотографируют объект с разных ракурсов и фотографии «накладываются» друг на друга в определенной последовательности кадров. А с помощью незамысловатого кода превращают в анимацию (пример объектов такой работы – рис.2), которая реагирует на движения компьютерной мыши. Так как этот метод достаточно популярен, то я не стал его реализовывать в коде, следовательно, не вижу смысла вставлять его в свою диссертацию, потому что эту информацию можно найти в интернете. Если сделать такие же фотографии, но на другом расстоянии между камерой и объектом, то можно достичь возможности вращения 3D объекта не только вокруг его оси, но еще и возможность отдалять и приближать его под разными углами. Но это уже более сложная и кропотливая работа. Поэтому для большей функциональности с 3D объектами чаще всего используют программы для 3D моделирования, о которых мы расскажем в следующей главе.

Рис2. Объект моделирования .

Б) Следующий способ создания трехмерных объектов - это моделирование 3D моделей с помощью разных программ, таких как 3d max, blender (рис.3) и другие. Данный процесс формирования 3D объектов представлен таким образом:

Рис3. Blender.

- сначала мы собираем наш объект в программе, для работы с трехмерными объектами (пример работы с рис.4). То есть либо загружаем уже готовую модель либо самостоятельно ее реализуем с помощью трехмерной графики и полигонов (в графике представляют собой треугольники – рис.5). Благодаря таким полигонам можно описать любой трехмерный объект, потому что любые три точки всегда лежат в одной плоскости. Таким образом, любую сложную поверхность можно описать таким методом. В каждой программе модель формируется разными способами, но в основном берется картинка или простой пример 3D объекта и растягивается таким образом, чтобы получить необходимую нам 3D модель;

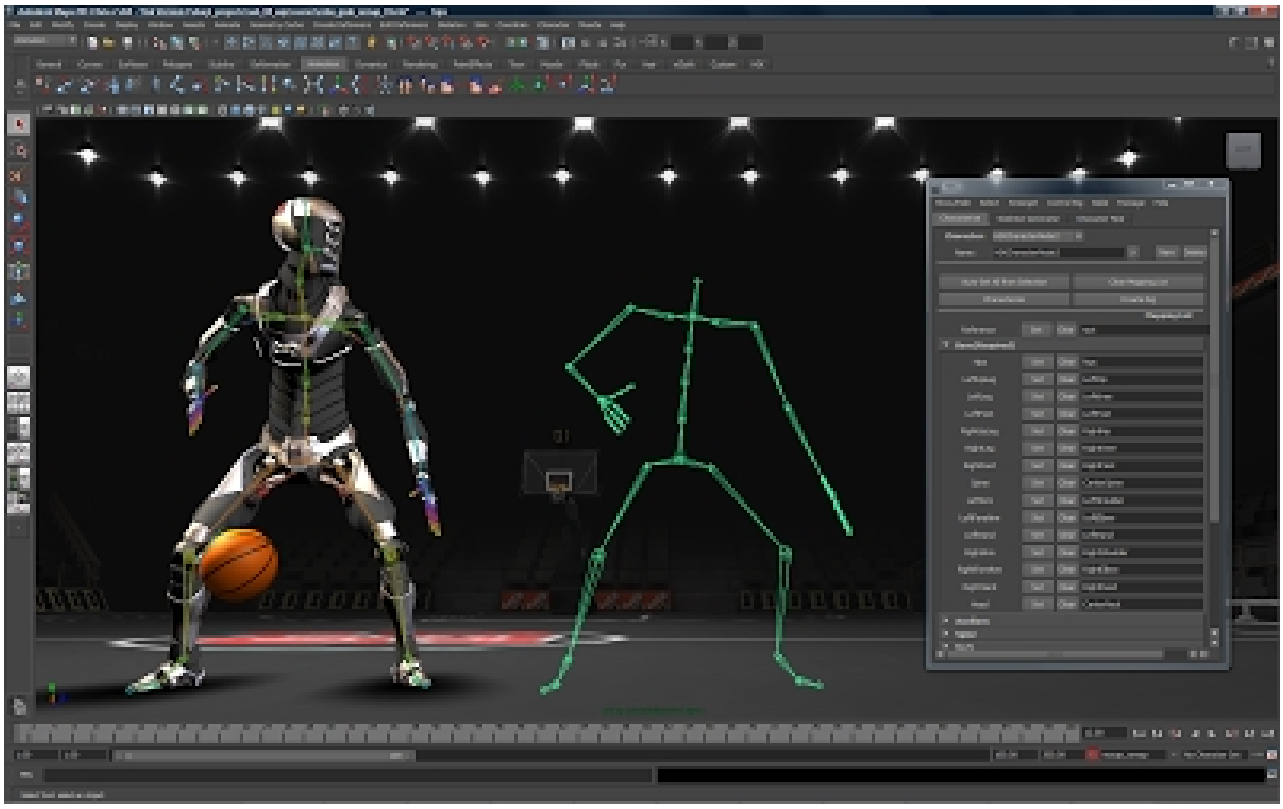

Рис4. Работа с программой 3D моделирования.

-после того как мы сделали нашу модель, мы изменяем формат файла (для примера - blender), который содержит ее описание в тот формат(collada), являющийся удобным для правильного подключения в страницу сайта, где будет требоваться работа с нашей моделью (пример модели представлен на рис.6.).

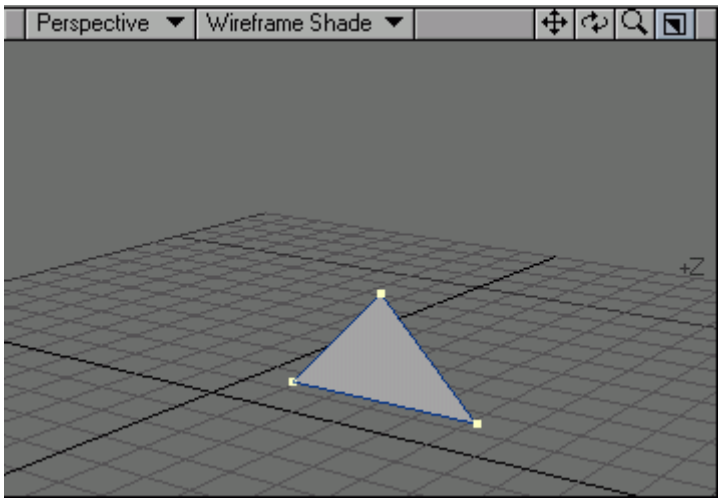

Рис5. Полигон.

Но проблема заключается в том, что в основном работа будет заключаться в возможностях, которые расписаны в коде. То есть мы сможем вращать, двигать модель, но корректировать ее мы не сможем (не будет возможности растягивать модель, менять ее внутри сценария страницы);

B) Сейчас я попытаюсь объяснить самый удобный вариант. Этот способ заключается в возможности формировать, изменять 3D объекты внутри сайта. И тут у нас так же есть два варианта:

-подключить готовую 3D модель за счет дополнительных библиотек и с возможностью корректировать 3D объекты;

- либо сформировать с нуля трехмерную модель внутри браузера и корректировать ее не закрывая сценарий (то есть модели не обязательно запускать к просмотру заранее, если они не нужны, ведь таким образом мы перегружаем страницу).

В первом случае, после того как мы подключили 3D модель к сайту, благодаря специальным приложениям появляется возможность двигать, менять модель, а так же погружаться в сам объект для просмотра его внутренней части. А во втором случае у нас еще есть возможность добавлять 3D модели в сцену. В

зависимости от вариантов моделей можно добавлять разные модели, но сохраняется возможность их корректирования. При этом старые примеры объектов не затираются, но и новые могут появиться только в том случае, когда у пользователя останется желание его сохранить или добавить в используемые трехмерные модели.

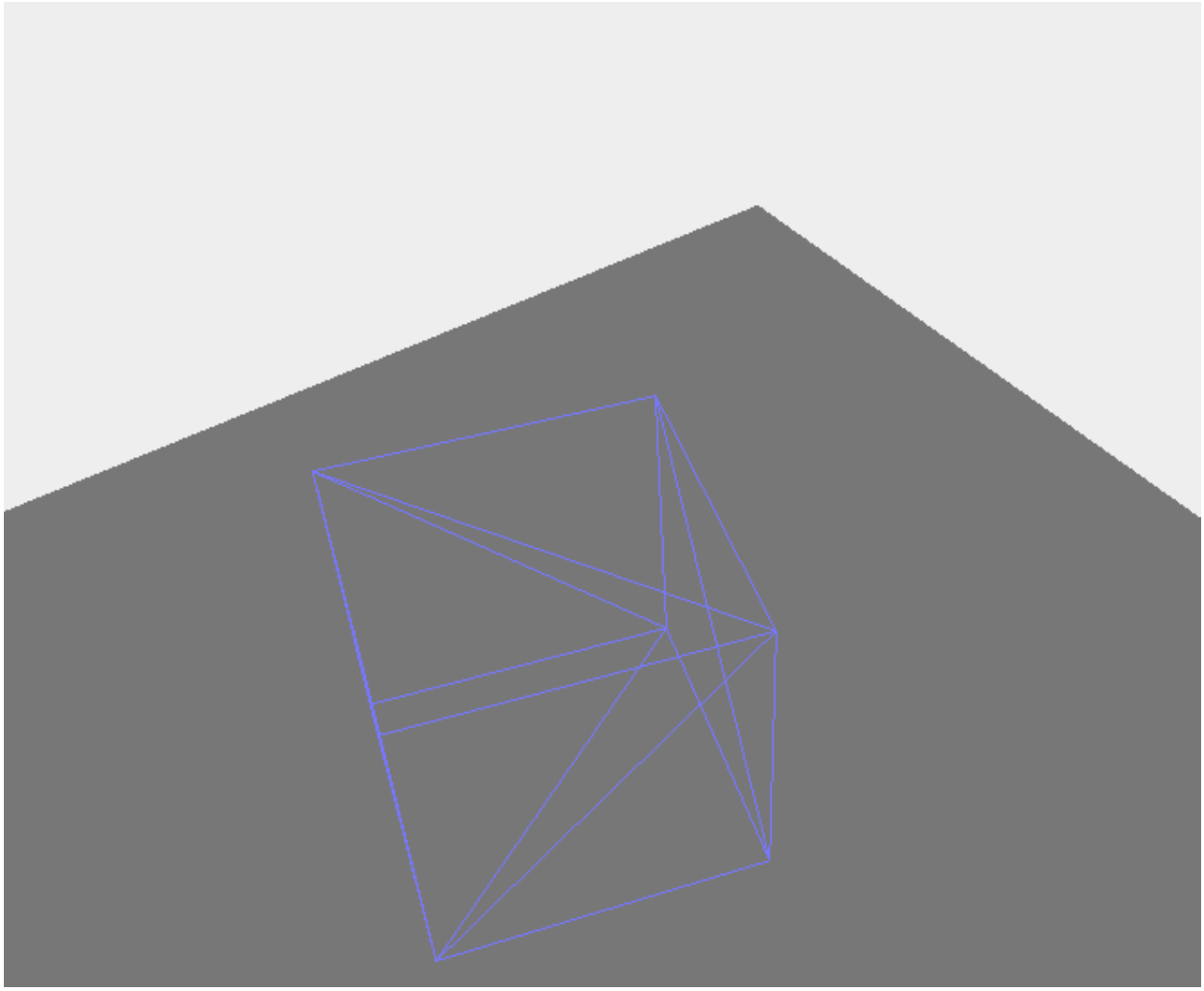

Рис6. Пример реализации трехмерного объекта в браузере.

В обоих вариантах мы используем две технологии: WebGL и Three.js(рис.7). В связи с такими технологиями появляется возможность работы с 3D моделями внутри сайта, web-приложения и прочего. То есть можем менять размеры, формы моделей или добавлять их в сайт. Дальше подробно

описываются эти технологии для общего понимания и дальнейшего правильного использования нашего приложения людьми.

Рис7. Three.js

Глава 3. Webgl и Three js

В этой главе мы подробно рассмотрим такие инструменты как WebGL и Three.js. Эти библиотеки позволят нам работать с графикой браузерного окна онлайн.

A) Для полного понимания WebGL, введем несколько определений:

- WebGL-программная библиотека для языка программирования JavaScript, позволяющая создавать на основе этого языка программирования интерактивную 3D графику, функционирующую в широком спектре совместимых с ней web-браузеров. За счет использования низкоуровневых средств поддержки OpenGL, а часть кода может выполняться непосредственно на видео картах[1];

1 определение с сайта https://ru.wikipedia.org/wiki/WebGL

- WebGL-это контекст элемента «canvas HTML», который обеспечивает API 3D графики без использования плагинов [2] .

Из всего этого следует, что данная библиотека может работать с графикой и объектами в трехмерном мире, а так же функционирует с браузерными системами. Иными словами, мы делаем 3D модель не на компьютере в рамках программы, работающей в операционной системе, а непосредственно в браузере, но в тоже самое время, формат этого файла не «флэш», к которому мы привыкли и не любим за то, что он постоянно тормозит. WebGL использует тег, который появился в HTML5, но рисуем уже в 3D, а не в 2D как привыкли.

Когда я впервые столкнулся с этой технологией, то мне было трудно в изучении и исследовании визуализации с помощью WebGL. Но благодаря книге (WebGL Beginner's Guide) посвященной этой библиотеке, множеству видео-уроков и прочим информационным форумам, я смог подключить сцену (так называемый canvas), камеру к сцене и добавлять модели и объекты в сцену с дальнейшей процедурой, включающую в себя работу с этими моделями. Но о библиотеке надо рассказать подробнее.

Рассказ пойдет о том, что появляется после того, как мы сформировали 3D-пространство. Сначала мы получаем рендереры, то есть некая сущность, отображенная на выбранную сцену с камерой. А под сценой, в данном случае, понимается пространство, в котором располагаются нужные объекты:

- элементы (еще их называют мешами);

- источники света, которые освещают элементы с какой-либо стороны.

---

2 определение с сайта http://www.dmitrylavrik.ru

Давайте представим для наглядности трехмерный прямоугольник, это и будет наша сцена (рис.8), также в ней за счет трех осей будет сформировано представление о трехмерном представлении этой сцены. На самой сцене будут располагаться различные элементы и эти элементы. Эти элементы и их местоположение, форма и другое зависит от камеры. Камера представляет собой место, с которого мы сейчас смотрим на сцену и куда мы смотрим.

Рис8. Сцена.

Из ходя из всего этого рендерер, для формирования изображения чего-то в сцену или иначе говоря на canvas, принимает два значения - сцену и камеру. И этого вполне достаточно, потому что основное представление пространства на тег canvas определяют именно они. Но в добавление скажу, что сцены и камеры могут отличаться друг от друга функциональностью и внешним видом. Например, есть простая камера (процесс работы камеры - рис.9), а есть плавающая, благодаря которой мы можем не только вращаться вокруг одной

точки, но еще и двигаться внутри самой сцены, так сказать будем материальной точкой, которая умеет двигаться в разные стороны. Так же есть примеры различных оформлений для сцены, в том числе формирование какого-то заднего плана, а это приведет к тому, что мы действительно получим какую-то приблизительно схожую к реальному миру картину всех вещей, а не просто бессмысленное и пустое пространство.

Но это еще не все. На сцене может быть еще что-то. Во первых на сцене может быть источник света - инструмент для освещения и создания специального цвета и тени объектов. Свет может быть направленным, точечным, рассеянным и так же влияющим на формы и качество теней, так что нужно быть очень внимательным и зачастую заранее хорошо обдумать, действительно ли данный функционал так нам нужен, не испортит ли он всей картины. Или наоборот улучшит гамму цветом и добавит свежую идею в описание сцены, которой так сильно не хватает многим изобретателям, художникам и другим людям.

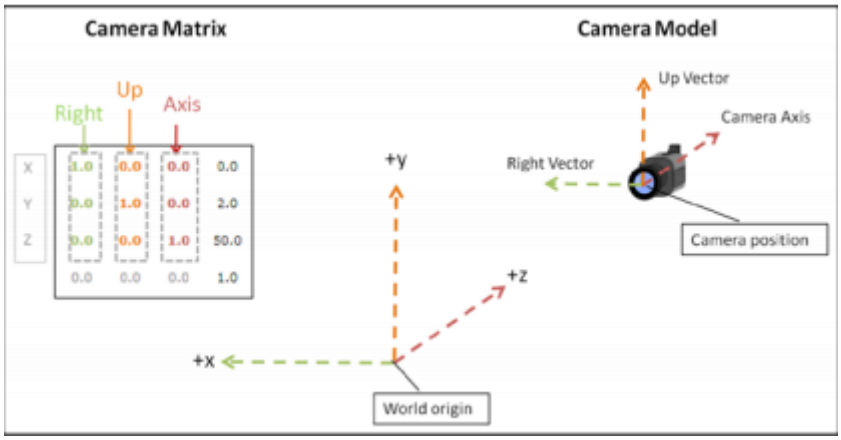

Рис9. Работа с камерой.

Еще есть такое понятие как меш, то есть элемент сцены, который состоит из геометрии и материала. И таким образом, можно представить любой объект

на нашей сцене, потому что любую трехмерную модель можно представить в виде определенной геометрии (вершины и ребра, которые их соединяет) и натянутым на нее материалом, так называемой кожей объекта. Исходя из этого, дадим определение геометрии и материала:

Геометрия - это множество точек вершин, которые при генерации соединяются между собой графическими примитивами;

Материал - это способ отображения и внешний вид элемента.

Иначе говоря, материал влияет на внешний вид элемента, того как он себя будет вести на свету, как отбрасывает тени и многое другое. А так же вместо цвета можно добавлять в материал картинку и размножать ее. Такой объект определяют как текстура - изображение, которое используется материалом для внешнего вида объекта. Для следующего шага нам потребуется следующее определение:

Шейдер – это компьютерная программа, предназначенная для исполнения процессорами видеокарты.

В WebGL меш описывают с помощью двух типов шейдеров:

- вершинные шейдеры;

- фрагментные шейдеры.

Вершинные шейдеры отвечают за координаты объекта, а фрагментные работают с текстурами и с частями растровых изображений.

С помощью шейдеров происходит процесс описания того, как именно будет сформирован трехмерный объект и какую форму он будет иметь. Шейдеры описывают 3D объекты в трехмерном пространстве с помощью полигональной сетки(совокупность вершин и связей между ними).

С помощью полигональной сетки формируется представление формы трехмерной модели, а чаще всего в качестве полигонов используют треугольники, но и с четырехугольниками тоже работают. В связи с правильной расстановкой полигонов и их характера строения (треугольник, четырехугольник), можно ускорить процесс формирования 3D объекта в трехмерном пространстве за счет уменьшения количества полигонов или редуплицирования полигональной сетки.

Таким образом, работа с 3D моделями находится в некотором пространстве, которое в целом называют сценой или «canvas». Для работы потребуется тег «canvas» и чтобы его сформировать воспользуемся не сложным методом, описывающим его в формате html документа внутри тега <body>:

```
<body>
    <canvas id="canvas-element-id" width="800" height="600">
    </canvas>
</body>
```

Простой код формирования сцены без дополнительных свойств целиком прописан в «Приложении 2».

После реализации сцены мы можем получить двухмерный квадрат внутри нее с помощью методов описанных ниже:

- initProgram(); // инициализирует программу подключения шейдеров цвета и шейдера вершин в сцену, которые отвечают за геометрию и материал объекта;

- initBuffers(); // описывает координаты вершин объекта;

-drawScene(); // отвечает за раскраску сцены;

- и другие методы, которые могут пригодиться в работе с данными проектами.

После знакомства с основами WebGl перейдем к описанию библиотеки Three.js.

Б) Three.js – легковесная кроссбраузерная библиотека JavaScript, используемая для создания и отображения анимированной компьютерной 3D графики при разработке web-приложений. Three.js скрипты могут использоваться совместно с элементами HTML (canvas), SVG (масштабированная векторная графика) и WebGL.

Как и все библиотеки их можно либо скачать на компьютер и подключить в код ссылкой на местоположение библиотеки в вашем компьютере либо ссылкой на исходные файлы в интернете.

В качестве примера, опишу фрагмент кода, используемый для формирования сферы(рис.10) в трехмерной сцене:

var scene = new THREE.Scene(); - подключение сцены.

var camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000); - Описываем камеру как объект с определенными атрибутами.

var geometry = new THREE.SphereGeometry(20, 20, 20); - определяем геометрию объекта.

var material = new THREE.MeshBasicMaterial({ color: 0x7777ff }); - определяем материал, из которого будет состоять объект.

var sphere = new THREE.Mesh(geometry, material); - Реализуем простой объект, в данном случае сферу, причем за основу берем геометрию,

описывающую размер и форму объекта, и материал, который отвечает за цвет трехмерного объекта.

scene.add(sphere); - добавляем в сцену сферу.

После описания методов позволяющих работать с простыми трехмерными моделями, перейдем к следующему пункту - сложные 3D объекты. Есть два варианта работы с такими объектами:

- формирование 3D моделей внутри сайта и потом их менять;

- реализовывать объекты в трехмерной графике в специальных программах, затем подключать их к странице с дальнейшей возможностью редактирования за счет функционала библиотеки Three.js.

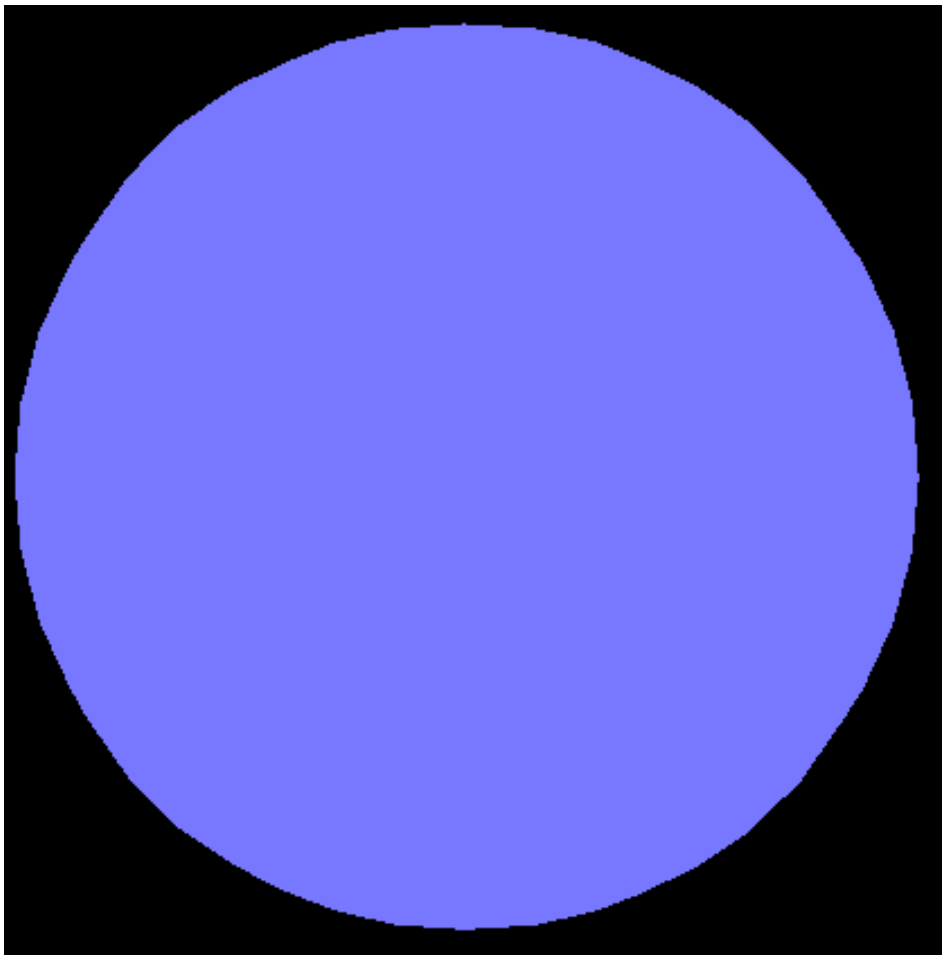

Рис10. Сфера.

Подключение готовых объектов к приложению, позволяет процесс загрузки. Для получения возможности формирования необходимого нам объекта, воспользуемся описание определенного загрузчика – представленного во фрагменте кода:

```
var loader  = THREE.STLLoader();

loader.addEventListener('load', function (event) {

    var geometry = event.content;

    scene.add (new THREE.Meshn (geometry));
```

```
} );

loader.load('./file storage/…/file.stl')
```

В этом фрагменте кода, так же описана геометрия(var geometry) подключаемого объекта, с которым мы решили работать. Таким образом, у нас есть возможность добавлять определенные модели и атрибуты, влияющие на графику 3D объектов в трехмерном пространстве.

Переходим к процессу реализации процедуры добавления трехмерных объектов пользователем в приложении, у которого есть возможности их изменения и подключения к виртуальной реальности. Эта процедура позволит людям получить возможность проверки правильности формирования трехмерной копии целой кости, которая сейчас повреждена у пациента.

Глава 4. Меню приложения

В процессе реализации приложения стало понятно, что пользователям будет удобней, если будет возможность загружать, редактировать и удалять объекты с помощью определенного меню (рис.11). Эта часть приложения будет способствовать быстрому освоению взаимного понимания между человеком и машиной. Меню реализуется за счет специальной библиотеки dat.gui.

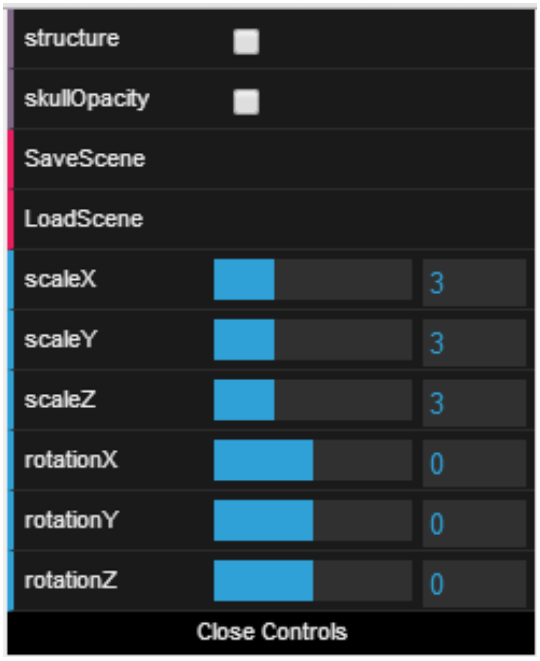

Рис11. Меню.

Фрагмент кода, который описывает оформление и функционал меню:

```
var gui = new dat.GUI();
gui.add(controls, 'structure');
gui.add(controls, 'skullOpacity').onChange(controls.changeSkullOpacity);
gui.add(controls, 'SaveScene');
gui.add(controls, 'LoadScene');
gui.add(controls, 'scaleX', 0, 10).onChange(controls.sclX);
gui.add(controls, 'scaleY', 0, 10).onChange(controls.sclY);
gui.add(controls, 'scaleZ', 0, 10).onChange(controls.sclZ);
gui.add(controls, 'rotationX', -4, 4).onChange(controls.rotateX);
gui.add(controls, 'rotationY', -4, 4).onChange(controls.rotateY);
gui.add(controls, 'rotationZ', -4, 4).onChange(controls.rotateZ);
```

Глава 5. Виртуальная реальность

После создания определенного интерфейса сайта с конкретным функционалом, осталось обратить внимание на последний из аспектов нашей задачи, который является формированием возможности просмотра сцены с объектами внутри виртуальной реальности. За реализацию этой процедуры отвечает Гвоздарев Илья Леонидович.

Так же хочется сказать, что возможность просмотра данного приложения в очках виртуальной реальности позволит проверять правильность структуры построенной трехмерной модели кости и ее правильной доработке, чтобы получить копию целой и невредимой кости, оригинал которой повредили и этот оригинал не подлежит восстановлению.

Заключение.

В конце исследования были выполнены все поставленные задачи. В силу заданных целей был сформирован план и действуя согласно его пунктам, мы смогли получить логичный и желаемый результат.

Итогом работы получили приложение с 3D сценой и меню, с помощью которых можно работать и добавлять 3D объекты. А так же наш продукт за счет библиотек WebGL и Three.js позволяет не сильно перегружать устройства пользователей для увеличения качества работы с приложением.

В результате нашего исследования данной задачи, было сформировано мнение о плюсах и минусах приложения.

Плюсы таковы:

- быстродействие системы, которые выполняется за счет быстрой загрузки всех методов кода, зависящей от видеокарты компьютера и быстрой обратной реакции на функционал посредством возможностей интернета;

-удобность использования библиотек трехмерной графики, на основе которых проводились исследования компьютерной визуализации и дальнейшее ее развитие;

- широкий спектр улучшения возможностей приложения.

В процессе реализации приложения формировались новые идеи для улучшения проекта, именно они и являются минусами нашей работы:

- процесс формирование трехмерных моделей увеличивает время загрузки объектов в сцену и ее решение лежит в нахождении упрощенного способа редупликации полигонов объектов;

- работа с данным приложением не совсем безопасна для людей. После долго нахождения в виртуальном трехмерном мире у людей появляется киберболезнь (Cybersickness) - это болезнь, симптомами которой являются головокружения, головные боли, тошнота и другие симптомы.

В результате исследования стало понятно, что люди желают как можно более простой продукт с реализованным приложением, который будет позволять им за короткое время использования получать нужный результат.

В будущем планируется обновления работы приложения, которые в первую очередь ликвидирует минусы этой версии, а так же позволит формировать базы моделей, с помощью которых появиться возможность повысить качество работы продукта, уровень хирургии в медицине и в других сферах жизни.

По ходу разработки web-приложения были изучены JavaScript, Webgl и Three.js. Так же были получены знания о способах и алгоритмах формирования объектов в трехмерном пространстве и дальнейшей работе с ними.

Список литературы.

1. " Webgl beginner's guide."Диего Кантор, Брэндон Джонс.
2. " Learning Three.js- The JavaScript 3D Library for WebGL." Джос Дирксэн.
3. https://ru.wikipedia.org/wiki/WebGL
4. https://www.dmirtylavrik.ru

Приложения.
Приложение1.(Код всей программы)

```
<!DOCTYPE html>

<html>

<head>
    <title>Example 09.07 - first person camera </title>
    <script type="text/javascript" src="three.js"></script>
    <script type="text/javascript" src="STLLoader.js"></script>
    <script type="text/javascript" src="stats.js"></script>
    <script type="text/javascript" src="dat.gui.js"></script>
    <script type="text/javascript" src="chroma.js"></script>
    <script type="text/javascript" src="FirstPersonControls.js"></script>
            <script type="text/javascript" src="Projector.js"></script>
            <script type="text/javascript" src="ColladaLoader.js"></script>
            <script type="text/javascript" src="ConvexGeometry.js"></script>
            <script type="text/javascript" src="ThreeBSP.js"></script>
            <script type="text/javascript" src="FileSaver.js"></script>
            <script type="text/javascript" src="OBJLoader.js"></script>
            <script type="text/javascript" src="helvetiker_bold.typeface.js"></
script>
    <script type="text/javascript"
src="helvetiker_regular.typeface.js"></script>

    <style>
        body {
            margin: 0;
            overflow: hidden;
        }
    </style>
</head>
<body>

<div id="Stats-output">
</div>
<div id="WebGL-output">
</div>
```

```
<script type="text/javascript">

    function init() {

                var vertexes = [];
                var objects = [];
                var parts = [];
                var vseRebra = [];
                var targetAnatomy;
                var savingObjects =[];
                var selection;
                var convex;
                var BSP1, BSP2, resultBSP, resultBSPtoMesh, loadconvex,
loadedConvex, objLoader;
                var index=0;
                var offset = new THREE.Vector3();
       var clock = new THREE.Clock();
                var stats = initStats();

       var scene = new THREE.Scene();

       var camera = new THREE.PerspectiveCamera(45, window.innerWidth /
window.innerHeight, 0.1, 1000);

                var webGLRenderer = new THREE.WebGLRenderer();
        webGLRenderer.setClearColor(new THREE.Color(0xEEEEEE, 1.0));
        webGLRenderer.setSize(window.innerWidth, window.innerHeight);
        webGLRenderer.shadowMapEnabled = true;

                var projector = new THREE.Projector();
        webGLRenderer.domElement.addEventListener('mousedown',
onDocumentMouseDown, false);
                webGLRenderer.domElement.addEventListener('mousemove',
onDocumentMouseMove, false);
                webGLRenderer.domElement.addEventListener('mouseup',
onDocumentMouseUp, false);


        camera.position.x = 100;
        camera.position.y = 10;
        camera.position.z = 10;
        camera.lookAt(new THREE.Vector3(0, 0, 0));

                var ambientLight = new THREE.AmbientLight(0x383838);
                ambientLight.intensity = 10;
        scene.add(ambientLight);

        var spotLight = new THREE.SpotLight(0xffffff);
        spotLight.position.set(100, 140, 130);
        spotLight.intensity = 1;
        scene.add(spotLight);

                var spotLight2 = new THREE.SpotLight(0xffffff);
        spotLight2.position.set(-100, -140, -130);
        spotLight2.intensity = 1;
        scene.add(spotLight2);
```

```
                plane = new THREE.Mesh(new THREE.PlaneBufferGeometry(500, 500,
8, 8), new THREE.MeshBasicMaterial({color: 0xff0000, side: THREE.DoubleSide}));
                plane.visible = false;
                scene.add(plane);



                sky = new THREE.Mesh( new THREE.SphereGeometry( 500, 100,
100 ), new THREE.MeshBasicMaterial( {
                map: THREE.ImageUtils.loadTexture( "322.jpg" ),
                side: THREE.DoubleSide
                } ) );
                scene.add(sky);


                var loader = new THREE.ColladaLoader();
        var skull;
        loader.load("skull.dae", function (result) {
            var geometry = result.scene.children[2].children[0].geometry;
                        var mat = new THREE.MeshLambertMaterial({color:
0xC0C0C0, side: THREE.DoubleSide});
            skull = new THREE.Mesh(geometry, mat);
            skull.scale.set(3, 3, 3);
                        skull.name ="skull";
                        skull.position.x = 20;
            scene.add(skull);
                        objects.push(skull);
        });

                var loaderCol = new THREE.ColladaLoader();
        var brain;
        loaderCol.load("brain.dae", function (result) {
                        var geometry =
result.scene.children[0].children[0].geometry;
                        var mat = new THREE.MeshLambertMaterial({color:
0xF08080, side: THREE.DoubleSide});
            brain = new THREE.Mesh(geometry, mat);
            brain.scale.set(3, 3, 3);
                        brain.name = "brain";
                        brain.position.x = 20;
            scene.add(brain);
                        objects.push(brain);
        });


                var loaderObj = new THREE.ObjectLoader();
                var part1, part2, part3, part4, part5, part6, part7;

                        loaderObj.load( "mozg1.json", function ( object ) {
                              var geometry = object.geometry;
                              var mat = new THREE.MeshLambertMaterial({color:
0x00ff00, side: THREE.DoubleSide});
                              part1 = new THREE.Mesh(geometry, mat);
                              part1.name = "part1";
                              part1.visible = false;
                              scene.add(part1);
```

```
                        part1.scale.set(3,3,3);
                        parts.push(part1);
                    } );

                    loaderObj.load( "mozg2.json", function ( object ) {
                        var geometry = object.geometry;
                        var mat = new THREE.MeshLambertMaterial({color:
0x00ff00, side: THREE.DoubleSide});
                        part2 = new THREE.Mesh(geometry, mat);
                        part2.name = "part2";
                        part2.visible = false;
                        scene.add(part2);
                        part2.scale.set(3,3,3);
                        parts.push(part2);
                    } );

                    loaderObj.load( "mozg3.json", function ( object ) {
                        var geometry = object.geometry;
                        var mat = new THREE.MeshLambertMaterial({color:
0x00ff00, side: THREE.DoubleSide});
                        part3 = new THREE.Mesh(geometry, mat);
                        part3.name = "part3";
                        part3.visible = false;
                        scene.add(part3);
                        part3.scale.set(3,3,3);
                        parts.push(part3);
                    } );

                    loaderObj.load( "mozg4.json", function ( object ) {
                        var geometry = object.geometry;
                        var mat = new THREE.MeshLambertMaterial({color:
0x00ff00, side: THREE.DoubleSide});
                        part4 = new THREE.Mesh(geometry, mat);
                        part4.name = "part4";
                        part4.visible = false;
                        scene.add(part4);
                        part4.scale.set(3,3,3);
                        parts.push(part4);
                    } );

                    loaderObj.load( "mozg5.json", function ( object ) {
                        var geometry = object.geometry;
                        var mat = new THREE.MeshLambertMaterial({color:
0x00ff00, side: THREE.DoubleSide});
                        part5 = new THREE.Mesh(geometry, mat);
                        part5.name = "part5";
                        part5.visible = false;
                        scene.add(part5);
                        part5.scale.set(3,3,3);
                        parts.push(part5);
                    } );

                    loaderObj.load( "mozg6.json", function ( object ) {
                        var geometry = object.geometry;
                        var mat = new THREE.MeshLambertMaterial({color:
0x00ff00, side: THREE.DoubleSide});
                        part6 = new THREE.Mesh(geometry, mat);
```

```
                        part6.name = "part6";
                        part6.visible = false;
                        scene.add(part6);
                        part6.scale.set(3,3,3);
                        parts.push(part6);
                    } );

                    loaderObj.load( "mozg7.json", function ( object ) {
                        var geometry = object.geometry;
                        var mat = new THREE.MeshLambertMaterial({color:
0x00ff00, side: THREE.DoubleSide});
                        part7 = new THREE.Mesh(geometry, mat);
                        part7.name = "part7";
                        part7.visible = false;
                        scene.add(part7);
                        part7.scale.set(3,3,3);
                        parts.push(part7);
                    } );


                var rebraloader = new THREE.ColladaLoader();
                var rebra;
                rebraloader.load("rebra.dae", function (result) {
                     var k=1;
                     for (var i=0; i<85; i+=2)
                     {
                        var geometry =
result.scene.children[i].children[0].geometry;
                        geometry.center();
                        var mat = new THREE.MeshLambertMaterial({color:
0xC0C0C0, side: THREE.DoubleSide});
                        rebra = new THREE.Mesh(geometry, mat);
                        rebra.scale.set(3,3,3);
                        rebra.name = "rebra"+k+""; k++;
                        scene.add(rebra);
                        objects.push(rebra);
                        vseRebra.push(rebra);
                     }

        });


                var options = {
                   size: 0.8,
                   height: 0.2,
                   weight: "normal",
                   font:"helvetiker",
                                style:"normal",
                   bevelEnabled: false
                };

                        var part1text1, part1text2, part1text3;

                        part1text1 = createMesh(new
THREE.TextGeometry("Occipital lobe:", options));
```

```
                            part1text2 = createMesh(new THREE.TextGeometry("
The occipital lobe is the visual processing center of the mammalian",
options));
                            part1text3 = createMesh(new THREE.TextGeometry("
brain containing most of the anatomical region of the visual cortex",
options));

                            var part2text1, part2text2, part2text3;
                            part2text1 = createMesh(new
THREE.TextGeometry("Left temporal lobe:", options));
                            part2text2 = createMesh(new THREE.TextGeometry("
The temporal lobe is involved in processing sensory input into derived meanings
for the", options));
                            part2text3 = createMesh(new THREE.TextGeometry("
appropriate retention of visual memories, language comprehension, and emotion
association", options));

                            var part3text1, part3text2, part3text3;
                            part3text1 = createMesh(new
THREE.TextGeometry("Right temporal lobe:", options));
                            part3text2 = createMesh(new THREE.TextGeometry("
The temporal lobe is involved in processing sensory input into derived meanings
for the", options));
                            part3text3 = createMesh(new THREE.TextGeometry("
appropriate retention of visual memories, language comprehension, and emotion
association", options));

                            var part4text1, part4text2, part4text3;
                            part4text1 = createMesh(new
THREE.TextGeometry("Pariental lobe:", options));
                            part4text2 = createMesh(new THREE.TextGeometry("
The parietal lobe integrates sensory information among various", options));
                            part4text3 = createMesh(new THREE.TextGeometry("
modalities, including spatial sense and navigation (proprioception)",
options));

                            var part5text1, part5text2, part5text3;
                            part5text1 = createMesh(new
THREE.TextGeometry("Hypophysis:", options));
                            part5text2 = createMesh(new THREE.TextGeometry("
The anterior pituitary synthesizes and secretes hormones", options));
                            part5text3 = createMesh(new THREE.TextGeometry("
Human growth hormone, Thyroid-stimulating hormone and other", options));

                            var part6text1, part6text2, part6text3;
                            part6text1 = createMesh(new
THREE.TextGeometry("Cerebellum:", options));
                            part6text2 = createMesh(new THREE.TextGeometry("
Is a region of the brain that plays an important role in motor control.",
options));
                            part6text3 = createMesh(new THREE.TextGeometry("
It may also be involved in some cognitive functions such as attention and
language", options));


                            var part7text1, part7text2, part7text3;
```

```
                              part7text1 = createMesh(new
THREE.TextGeometry("Frontal lobe:", options));
                              part7text2 = createMesh(new THREE.TextGeometry("
The dopamine system is associated with reward, attention,", options));
                              part7text3 = createMesh(new THREE.TextGeometry("
short-term memory tasks, planning, and motivation.", options));


                  var camControls = new THREE.FirstPersonControls(camera,
webGLRenderer.domElement);
         camControls.lookSpeed = 0.4;
         camControls.movementSpeed = 20;
         camControls.noFly = true;
         camControls.lookVertical = true;
         camControls.constrainVertical = true;
         camControls.verticalMin = 0.0;
         camControls.verticalMax = 3.0;
         camControls.lon = -150;
         camControls.lat = 120;


        var step = 0;

         var mesh;


         function setCamControls() {

         }

                   var controls = new function () {

                        this.structure = false;
                        this.skullOpacity = false;
                        this.sclX = function () {
                             targetAnatomy.scale.x = controls.scaleX;
                             if (targetAnatomy.name == "brain") {for (var i=0;
i<parts.length; i++) parts[i].scale.x = targetAnatomy.scale.x;}
             };

                        this.sclY = function () {
                             targetAnatomy.scale.y = controls.scaleY;
                             if (targetAnatomy.name == "brain") {for (var i=0;
i<parts.length; i++) parts[i].scale.y = targetAnatomy.scale.y;}
             };

                        this.sclZ = function () {
                             targetAnatomy.scale.z = controls.scaleZ;
                             if (targetAnatomy.name == "brain") {for (var i=0;
i<parts.length; i++) parts[i].scale.z = targetAnatomy.scale.z;}
             };

                        this.rotateX = function () {
                             targetAnatomy.rotation.x = controls.rotationX;
                             if (targetAnatomy.name == "brain") {for (var i=0;
i<parts.length; i++) parts[i].rotation.x = targetAnatomy.rotation.x;}
             };
```

```
                        this.rotateY = function () {
                              targetAnatomy.rotation.y = controls.rotationY;
                              if (targetAnatomy.name == "brain") {for (var i=0;
i<parts.length; i++) parts[i].rotation.y = targetAnatomy.rotation.y;}
                  };

                        this.rotateZ = function () {
                              targetAnatomy.rotation.z = controls.rotationZ;
                              if (targetAnatomy.name == "brain") {for (var i=0;
i<parts.length; i++) parts[i].rotation.z = targetAnatomy.rotation.z;}
                  };

                        this.scaleX = 3;
                        this.scaleY = 3;
                        this.scaleZ = 3;
                        this.rotationX = 0;
                        this.rotationY = 0;
                        this.rotationZ = 0;

                        this.SaveScene = function () {
                              var k=0;
                              for (var i=0; i<objects.length; i++) {
                                    savingObjects[k] = objects[i].position.x;
                                    savingObjects[k+1] = objects[i].position.y;
                                    savingObjects[k+2] = objects[i].position.z;
                                    savingObjects[k+3] = objects[i].scale.x;
                                    savingObjects[k+4] = objects[i].scale.y;
                                    savingObjects[k+5] = objects[i].scale.z;
                                    savingObjects[k+6] = objects[i].rotation.x;
                                    savingObjects[k+7] = objects[i].rotation.y;
                                    savingObjects[k+8] = objects[i].rotation.z;
                                    k+=9;
                              }
                        };

                        this.LoadScene = function () {
                              var k=0;
                              for (var i=0; i<objects.length; i++) {
                                    objects[i].position.x = savingObjects[k];
                                    objects[i].position.y = savingObjects[k+1];
                                    objects[i].position.z = savingObjects[k+2];
                                    objects[i].scale.x = savingObjects[k+3];
                                    objects[i].scale.y = savingObjects[k+4];
                                    objects[i].scale.z = savingObjects[k+5];
                                    objects[i].rotation.x = savingObjects[k+6];
                                    objects[i].rotation.y = savingObjects[k+7];
                                    objects[i].rotation.z = savingObjects[k+8];
                                    k+=9;
                              }
                        };

                        this.changeSkullOpacity = function () {
                              if (controls.skullOpacity == true)
{ skull.material.transparent = true; skull.material.opacity=0.6;}
                              if (controls.skullOpacity == false)
{ skull.material.transparent = false;}
```

```
                    }

        };



                var gui = new dat.GUI();
                gui.add(controls, 'structure');
                gui.add(controls,
'skullOpacity').onChange(controls.changeSkullOpacity);
                gui.add(controls, 'SaveScene');
                gui.add(controls, 'LoadScene');
                gui.add(controls, 'scaleX', 0, 10).onChange(controls.sclX);
                gui.add(controls, 'scaleY', 0, 10).onChange(controls.sclY);
                gui.add(controls, 'scaleZ', 0, 10).onChange(controls.sclZ);
                gui.add(controls, 'rotationX', -4,
4).onChange(controls.rotateX);
                gui.add(controls, 'rotationY', -4,
4).onChange(controls.rotateY);
                gui.add(controls, 'rotationZ', -4,
4).onChange(controls.rotateZ);

        render();


        function render() {
            stats.update();
            var delta = clock.getDelta();

            camControls.update(delta);
            requestAnimationFrame(render);
            webGLRenderer.render(scene, camera)
        }

                function createMesh(geom) {

            var meshMaterial = new THREE.MeshPhongMaterial({
                specular: 0xffffff,
                color: 0xeeffff,
                shininess: 100,
                metal: true
            });
            var plane = THREE.SceneUtils.createMultiMaterialObject(geom,
[meshMaterial]);
            return plane;
        }

                var projector = new THREE.Projector();
        var tube;

        function onDocumentMouseDown(event) {
                            var vector = new THREE.Vector3(( event.clientX /
window.innerWidth ) * 2 - 1, -( event.clientY / window.innerHeight ) * 2 + 1,
1);
                            vector.unproject(camera);
```

```
                              var raycaster = new
THREE.Raycaster(camera.position, vector.sub(camera.position).normalize());
                              var intersects =
raycaster.intersectObjects(objects);
                              if (intersects.length > 0) {
                                    targetAnatomy = intersects[0].object;
                                    camControls.enabled = false;
                                    selection = intersects[0].object;
                                    var intersects =
raycaster.intersectObject(plane);

            offset.copy(intersects[0].point).sub(plane.position);

                              }

        }

                  var selectedObject = null;

                  function onDocumentMouseMove(event) {
                        event.preventDefault();

                        var vector = new THREE.Vector3(( event.clientX /
window.innerWidth ) * 2 - 1, -( event.clientY / window.innerHeight ) * 2 + 1,
1);
                        vector.unproject(camera);

                        var raycaster = new THREE.Raycaster(camera.position,
vector.sub(camera.position).normalize());

                        if (controls.structure) {
                              var intersects = raycaster.intersectObjects(parts);
                              if (intersects.length > 0) {
                                    brain.material.transparent = true;
                                    brain.material.opacity = 0.2;
                                    intersects[0].object.visible = true;
                                    if (intersects[0].object.name == "part1")
                                    {
                                          part1text1.position.x =
(intersects[0].object.position.x + camera.position.x) / 2;
                                          part1text1.position.y =
(intersects[0].object.position.y + camera.position.y) / 2;
                                          part1text1.position.z =
(intersects[0].object.position.z + camera.position.z) / 2;
                                          part1text2.position.x =
part1text1.position.x;
                                          part1text2.position.y =
part1text1.position.y-1;
                                          part1text2.position.z =
part1text1.position.z;
                                          part1text3.position.x =
part1text2.position.x;
                                          part1text3.position.y =
part1text2.position.y-1;
                                          part1text3.position.z =
part1text2.position.z;
                                          part1text1.scale.set(0.5,0.5,0.5);
```

```
                                        part1text2.scale.set(0.5,0.5,0.5);
                                        part1text3.scale.set(0.5,0.5,0.5);
                                        part1text1.lookAt(camera.position);
                                        part1text2.lookAt(camera.position);
                                        part1text3.lookAt(camera.position);
                                        scene.add(part1text1);
scene.add(part1text2); scene.add(part1text3);
                                    }

                                    if (intersects[0].object.name == "part2")
                                    {
                                        part2text1.position.x =
(intersects[0].object.position.x + camera.position.x) / 2;
                                        part2text1.position.y =
(intersects[0].object.position.y + camera.position.y) / 2;
                                        part2text1.position.z =
(intersects[0].object.position.z + camera.position.z) / 2;
                                        part2text2.position.x =
part2text1.position.x;
                                        part2text2.position.y =
part2text1.position.y-1;
                                        part2text2.position.z =
part2text1.position.z;
                                        part2text3.position.x =
part2text2.position.x;
                                        part2text3.position.y =
part2text2.position.y-1;
                                        part2text3.position.z =
part2text2.position.z;
                                        part2text1.scale.set(0.5,0.5,0.5);
                                        part2text2.scale.set(0.5,0.5,0.5);
                                        part2text3.scale.set(0.5,0.5,0.5);
                                        part2text1.lookAt(camera.position);
                                        part2text2.lookAt(camera.position);
                                        part2text3.lookAt(camera.position);
                                        scene.add(part2text1);
scene.add(part2text2); scene.add(part2text3);
                                    }

                                    if (intersects[0].object.name == "part3")
                                    {
                                        part3text1.position.x =
(intersects[0].object.position.x + camera.position.x) / 2;
                                        part3text1.position.y =
(intersects[0].object.position.y + camera.position.y) / 2;
                                        part3text1.position.z =
(intersects[0].object.position.z + camera.position.z) / 2;
                                        part3text2.position.x =
part3text1.position.x;
                                        part3text2.position.y =
part3text1.position.y-1;
                                        part3text2.position.z =
part3text1.position.z;
                                        part3text3.position.x =
part3text2.position.x;
                                        part3text3.position.y =
part3text2.position.y-1;
```

```
                                        part3text3.position.z =
part3text2.position.z;
                                        part3text1.scale.set(0.5,0.5,0.5);
                                        part3text2.scale.set(0.5,0.5,0.5);
                                        part3text3.scale.set(0.5,0.5,0.5);
                                        part3text1.lookAt(camera.position);
                                        part3text2.lookAt(camera.position);
                                        part3text3.lookAt(camera.position);
                                        scene.add(part3text1);
scene.add(part3text2); scene.add(part3text3);
                                    }

                                    if (intersects[0].object.name == "part4")
                                    {
                                        part4text1.position.x =
(intersects[0].object.position.x + camera.position.x) / 2;
                                        part4text1.position.y =
(intersects[0].object.position.y + camera.position.y) / 2;
                                        part4text1.position.z =
(intersects[0].object.position.z + camera.position.z) / 2;
                                        part4text2.position.x =
part4text1.position.x;
                                        part4text2.position.y =
part4text1.position.y-1;
                                        part4text2.position.z =
part4text1.position.z;
                                        part4text3.position.x =
part4text2.position.x;
                                        part4text3.position.y =
part4text2.position.y-1;
                                        part4text3.position.z =
part4text2.position.z;
                                        part4text1.scale.set(0.5,0.5,0.5);
                                        part4text2.scale.set(0.5,0.5,0.5);
                                        part4text3.scale.set(0.5,0.5,0.5);
                                        part4text1.lookAt(camera.position);
                                        part4text2.lookAt(camera.position);
                                        part4text3.lookAt(camera.position);
                                        scene.add(part4text1);
scene.add(part4text2); scene.add(part4text3);
                                    }

                                    if (intersects[0].object.name == "part5")
                                    {
                                        part5text1.position.x =
(intersects[0].object.position.x + camera.position.x) / 2;
                                        part5text1.position.y =
(intersects[0].object.position.y + camera.position.y) / 2;
                                        part5text1.position.z =
(intersects[0].object.position.z + camera.position.z) / 2;
                                        part5text2.position.x =
part5text1.position.x;
                                        part5text2.position.y =
part5text1.position.y-1;
                                        part5text2.position.z =
part5text1.position.z;
```

```
                                        part5text3.position.x =
part5text2.position.x;
                                        part5text3.position.y =
part5text2.position.y-1;
                                        part5text3.position.z =
part5text2.position.z;
                                        part5text1.scale.set(0.5,0.5,0.5);
                                        part5text2.scale.set(0.5,0.5,0.5);
                                        part5text3.scale.set(0.5,0.5,0.5);
                                        part5text1.lookAt(camera.position);
                                        part5text2.lookAt(camera.position);
                                        part5text3.lookAt(camera.position);
                                        scene.add(part5text1);
scene.add(part5text2); scene.add(part5text3);
                                    }

                                    if (intersects[0].object.name == "part6")
                                    {
                                        part6text1.position.x =
(intersects[0].object.position.x + camera.position.x) / 2;
                                        part6text1.position.y =
(intersects[0].object.position.y + camera.position.y) / 2;
                                        part6text1.position.z =
(intersects[0].object.position.z + camera.position.z) / 2;
                                        part6text2.position.x =
part6text1.position.x;
                                        part6text2.position.y =
part6text1.position.y-1;
                                        part6text2.position.z =
part6text1.position.z;
                                        part6text3.position.x =
part6text2.position.x;
                                        part6text3.position.y =
part6text2.position.y-1;
                                        part6text3.position.z =
part6text2.position.z;
                                        part6text1.scale.set(0.5,0.5,0.5);
                                        part6text2.scale.set(0.5,0.5,0.5);
                                        part6text3.scale.set(0.5,0.5,0.5);
                                        part6text1.lookAt(camera.position);
                                        part6text2.lookAt(camera.position);
                                        part6text3.lookAt(camera.position);
                                        scene.add(part6text1);
scene.add(part6text2); scene.add(part6text3);
                                    }

                                    if (intersects[0].object.name == "part7")
                                    {
                                        part7text1.position.x =
(intersects[0].object.position.x + camera.position.x) / 2;
                                        part7text1.position.y =
(intersects[0].object.position.y + camera.position.y) / 2;
                                        part7text1.position.z =
(intersects[0].object.position.z + camera.position.z) / 2;
                                        part7text2.position.x =
part7text1.position.x;
```

```
                                        part7text2.position.y =
part7text1.position.y-1;
                                        part7text2.position.z =
part7text1.position.z;
                                        part7text3.position.x =
part7text2.position.x;
                                        part7text3.position.y =
part7text2.position.y-1;
                                        part7text3.position.z =
part7text2.position.z;
                                        part7text1.scale.set(0.5,0.5,0.5);
                                        part7text2.scale.set(0.5,0.5,0.5);
                                        part7text3.scale.set(0.5,0.5,0.5);
                                        part7text1.lookAt(camera.position);
                                        part7text2.lookAt(camera.position);
                                        part7text3.lookAt(camera.position);
                                        scene.add(part7text1);
scene.add(part7text2); scene.add(part7text3);
                                    }

                                    if (selectedObject && (selectedObject.name !=
intersects[0].object.name)) {
                                            selectedObject.visible = false;
                                            if (selectedObject.name == "part1")
{ scene.remove(part1text1); scene.remove(part1text2); scene.remove(part1text3);
}
                                            if (selectedObject.name == "part2")
{ scene.remove(part2text1); scene.remove(part2text2); scene.remove(part2text3);
}
                                            if (selectedObject.name == "part3")
{ scene.remove(part3text1); scene.remove(part3text2); scene.remove(part3text3);
}
                                            if (selectedObject.name == "part4")
{ scene.remove(part4text1); scene.remove(part4text2); scene.remove(part4text3);
}
                                            if (selectedObject.name == "part5")
{ scene.remove(part5text1); scene.remove(part5text2); scene.remove(part5text3);
}
                                            if (selectedObject.name == "part6")
{ scene.remove(part6text1); scene.remove(part6text2); scene.remove(part6text3);
}
                                            if (selectedObject.name == "part7")
{ scene.remove(part7text1); scene.remove(part7text2); scene.remove(part7text3);
}
                                            selectedObject.material.transparent
= false; selectedObject=null;
                                        }
                                    selectedObject = intersects[0].object;
                                }
                                else if (selectedObject) {
                                            selectedObject.visible = false;
                                            brain.material.transparent = false;
                                            if (selectedObject.name == "part1")
{ scene.remove(part1text1); scene.remove(part1text2); scene.remove(part1text3);
}
```

```
                                        if (selectedObject.name == "part2")
{ scene.remove(part2text1); scene.remove(part2text2); scene.remove(part2text3);
}
                                        if (selectedObject.name == "part3")
{ scene.remove(part3text1); scene.remove(part3text2); scene.remove(part3text3);
}
                                        if (selectedObject.name == "part4")
{ scene.remove(part4text1); scene.remove(part4text2); scene.remove(part4text3);
}
                                        if (selectedObject.name == "part5")
{ scene.remove(part5text1); scene.remove(part5text2); scene.remove(part5text3);
}
                                        if (selectedObject.name == "part6")
{ scene.remove(part6text1); scene.remove(part6text2); scene.remove(part6text3);
}
                                        if (selectedObject.name == "part7")
{ scene.remove(part7text1); scene.remove(part7text2); scene.remove(part7text3);
}
                                            selectedObject.material.transparent =
false;
                                            selectedObject = null;
                                    }

                            }

                            if (selection) {
                                  var intersects = raycaster.intersectObject(plane);

                  selection.position.copy(intersects[0].point.sub(offset));
                                  if (selection.name == "skull")
{brain.position.copy(skull.position); brain.position.y+=8;}
                                  for (var i=0; i<parts.length; i++)
parts[i].position.copy(brain.position);   //чтобы части двигались вместе с
мозгом

                            } else
                            {
                                    var intersects =
raycaster.intersectObjects(objects);
                                  if (intersects.length > 0) {

                  plane.position.copy(intersects[0].object.position);
                                      plane.lookAt(camera.position);
                                  }

                            }

                      }

                      function onDocumentMouseUp(event) {
                            camControls.enabled = true;
                            selection = null;
                      }

              function initStats() {

                  var stats = new Stats();
```

```
            stats.setMode(0); // 0: fps, 1: ms

            stats.domElement.style.position = 'absolute';
            stats.domElement.style.left = '0px';
            stats.domElement.style.top = '0px';

            document.getElementById("Stats-
output").appendChild(stats.domElement);

            return stats;
        }
    }

                window.onload = init;
</script>
</body>
</html>
```

Приложение2.(Код программы с трехмерной сценой)

```
<!DOCTYPE html>
<html>
       <head>
            <title> …</title>
           <style type="text/css">
                  canvas {border: 2px dotted blue;}
           </style>
        </head>
       <body>
           <canvas id="canvas-element-id" width="800" height="600">
           </canvas>
       </body>
</html>
```