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. Реализация взаимодействия с виртуальными объектами»

«Допущен к защите»	Квалификационная работа на соискание степени магистра наук по направлению 02.04.01 Математика и компьютерные науки Магистерская программа «Математическая кибернетика» студента гр. МКМ-240602 Гвоздарева Ильи Леонидовича
_____ «___»_____2016 г.	Научный руководитель Авербух Владимир Лазаревич, доцент, кандидат технических наук

Екатеринбург
2016

РЕФЕРАТ

Гвоздарева И.Л. СИСТЕМА УДАЛЕННОЙ ВИЗУАЛИЗАЦИИ ТРЕХМЕРНЫХ ОБЪЕКТОВ С ИСПОЛЬЗОВАНИЕМ СРЕД ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ. РЕАЛИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ С ВИРТУАЛЬНЫМИ ОБЪЕКТАМИ,

дипломная работа: стр. 40, рис. 19, табл. 4.

Ключевые слова: ВИЗУАЛИЗАЦИЯ, РЕНДЕРИНГ, THREE.JS, WEB, КАМЕРА, ОБЪЕКТ, 3D-СЦЕНА, ОЧКИ ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ.

Объект исследования – виртуальная графическая трехмерная сцена.

Цель работы – разработка и программирование трехмерной сцены, инструментария пользователя и средств управления анатомическими объектами используя методы виртуальной реальности.

В результате проделанной работы, был реализован необходимый инструментарий и средства манипуляции анатомическими объектами, загруженными в сцену из внешних источников.

Была реализована возможность просмотра сцены в очках виртуальной реальности в окне браузера.

В перспективе предполагается, что анализ моделей будет проводиться либо медиком, ведущим исследование и/или лечение больного, либо специалистом, подготовляющим проведение 3D-принтинга для физической визуализации и иных объектов организма.

СОДЕРЖАНИЕ

Введение

Идея создания изображений очень стара, но мысль о том, что можно получать изображения внутренних структур живого человеческого тела, стала реальностью с открытием рентгеновских лучей в 1895 г. За последующие 50 лет это открытие не слишком далеко ушло от эмпирического уровня и, лишь начиная приблизительно с 1950 г. с появлением других методов, таких как радиоизотопная и ультразвуковая визуализация, термография, рентгеновская и ЯМР компьютерная томография, в этом направлении были достигнуты некоторые успехи и появились признаки формирования научного подхода. Несомненно, этому помогло одновременное развитие общих принципов визуализации в технике связи – развитие, нашедшее широкий отклик в промышленных и военных приложениях процессов записи, передачи и отображения графической информации.

Современные методы визуализации: УЗИ (ультразвуковое исследование), КТ (рентгеновская компьютерная томография) и МРТ (магнитно-резонансная томография) – дают возможность прижизненного изучения структуры органов, анатомических объектов организма, а также позволяют получить информационные модели изучаемых объектов, которые затем могут быть

загружены в компьютер для дальнейшей обработки, визуализации или 3d-печати.

Задача компьютерной графики – визуализация, т.е. создание изображения. Визуализация выполняется исходя из описания (модели) того, что нужно отображать. Существует много методов и алгоритмов визуализации, которые различаются между собой в зависимости от того, что и как отображать (например, отображение графика функции, диаграммы, карты или схемы или отображение реальной трехмерной сцены в играх, художественных и мультипликационных фильмах, в системах архитектурного проектирования). Визуализация — один из наиболее важных разделов в компьютерной графике, и на практике он тесным образом связан с остальными. Обычно программные пакеты трёхмерного моделирования и анимации включают в себя также и функцию рендеринга. Существуют отдельные программные продукты, выполняющие рендеринг: RenderMan, YafaRay, V-Ray, LuxRender и другие, а также различные технологии визуализации: растеризация, ray casting, трассировка лучей.

Задача компьютерной визуализации тесно связана с задачей человеко-компьютерного взаимодействия – улучшение взаимодействия между человеком и компьютером, делая компьютеры более удобными и восприимчивыми к потребностям пользователей.

Цель данной работы – используя современные подходы и технологии в компьютерной графике, теории визуализации и человеко-компьютерного взаимодействия, создать виртуальную трехмерную сцену. Используя готовые 3d-модели анатомических объектов (костей скелета и головной мозг), полученные на приборах современной компьютерной визуализации, загружать их в сцену для дальнейшего их изучения в среде виртуальной реальности и манипуляции ими.

В перспективе предполагается, что анализ моделей будет проводиться либо медиком, ведущим исследование и/или лечение больного, либо специалистом, подготавливающим проведение 3D-принтинга для физической визуализации и иных объектов организма. Результаты КТ, ЯМРТ и, теоретически, любого медицинского исследования с визуализацией традиционно можно получить в формате DICOM (отраслевой стандарт создания, хранения, передачи и визуализации медицинских изображений и документов обследованных пациентов). Затем с помощью специального программного обеспечения, например 3D Slicer, вы можете преобразовать результат исследования в любой известный формат с данными о модели.

Постановка задачи

В задачи данной работы входит:

1. Изучение средств компьютерной визуализации.
2. Разработка инструментария пользователя и средств управления анатомическими объектами.
3. Разработка программного обеспечения и алгоритмов для обеспечения работы объектов сцены и корректной визуализации.
4. Освоение очков виртуальной реальности и их использование для создания зрительного эффекта присутствия при взаимодействии с анатомическими объектами.
5. Создание интерфейса для упрощения работы пользователя с 3D-сценой.

WebGL и Three.js

В последние пару лет браузеры стали более мощными и способны использовать платформы для реализации комплексных приложений и графики. Большинство из них, тем не менее, являются стандартными двумерными графиками. Большинство современных браузеров приняли WebGL, который позволяет не только создавать 2D-приложения и графики в браузере, но и создавать красивые 3D-приложения, использующие возможности графического процессора.

Программирование WebGL напрямую, однако, является очень сложным. Вы должны знать внутренние детали WebGL и изучать сложный язык затенения, чтобы получить максимальную отдачу от WebGL. Three.js обеспечивает очень простой в использовании JavaScript API вокруг особенностей WebGL, так что вы можете создавать красивые 3D-графики без необходимости изучать WebGL в деталях.

Как мы и сказали, современные браузеры постепенно приобретают все более мощные функции, которые могут быть доступны непосредственно из JavaScript. Вы можете легко добавлять видео и аудио в новые теги HTML5 и создавать интерактивные компоненты за счет использования холста HTML5.

Вместе с HTML5 современные браузеры также начали поддерживать WebGL. С помощью WebGL вы можете напрямую использовать ресурсы видеокарты и создавать высокопроизводительные двумерные и трехмерные компьютерные графики. Программирование WebGL происходит непосредственно из JavaScript, создание и анимация 3D сцены является очень сложным и подверженным ошибкам процессом. Three.js это библиотека, которая делает этот процесс намного проще. В следующем списке приведены некоторые из вещей, работа с которыми легко возможна в Three.js:

• Создание простых и сложных 3D геометрий

• Анимационные и движущиеся объекты в 3D-сцене

• Применение текстур и материалов для ваших объектов

• Использование различных источников света для освещения сцены

• Загрузка объектов, 3D-моделей и целых сцен

• Добавление расширенной постобработки эффектов

• Работа с вашими собственными шейдерами

• Создание облака точек (спрайты)

Основная библиотека, используемая для создания и обработки нашей сцены – Three.js (3D Javascript Library):

Three.js предоставляет большое количество функций и API, которые можно использовать для создания 3D-сцены прямо в браузере. Three.js - это JavaScript API для визуализации интерактивных 2D и 3D график, которые встраиваются внутри элемента HTML <Canvas>.

Рис. . Поддерживаемые браузеры

Поддерживаемые Визуализаторы (Renderers):

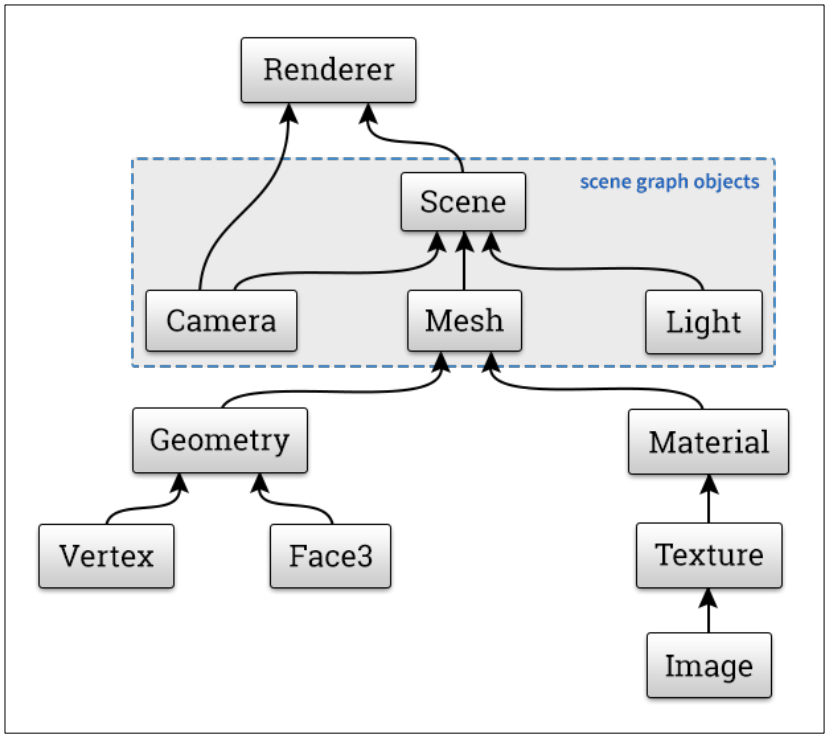
WebGL, <canvas>, <svg>, CSS3D / DOM и другие

Поддерживаемые технологии:

Scenes, Cameras, Geometry, 3D Model Loaders, Lights, Materials, Shaders, Particles, Animation, Math Utilities.

Структура сцены:

Рис. . Структура сцены WebGL



Настройка Web-сервера

Для большинства простых скриптов, использующих простые модели и внутренние ресурсы, достаточно просто открыть пример в браузере. Но когда мы начинаем подгрузку внешних ресурсов, таких как модели или текстуры изображений, просто открыть файл HTML недостаточно. В этом случае нам нужен локальный веб-сервер, чтобы убедиться, что внешние ресурсы загружены правильно.

Есть несколько различных способов решения данной проблемы:
можно создать простой локальный веб-сервер для тестирования, либо, если вы не можете создать локальный веб-сервер, но используете браузеры Chrome или Firefox, можно отключить определенные функции безопасности и разрешить загрузку из файлов в настройках запуска браузера.

Есть много различных способов создать и настроить локальный веб-сервер. В данной работе был использован простой портативный локальный веб-сервер Mongoose. Если вы используете Windows, достаточно скопировать исполняемый файл Mongoose-free-6.4.exe в каталог, содержащий ваши примеры и дважды нажать на исполняемый файл, чтобы запустить веб-браузер, где предлагается каталог папки из которой был запущен файл.

Рис. . Каталог файлов при запуске локального веб-сервера

Создание сцены, объектов и ее визуализация

Определение сцены:

```
var scene = new THREE.Scene();
```

Определение камеры. Камера определяет то, что пользователь видит в момент визуализации сцены:

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

Определение визуализатора. Объект визуализатор несет ответственность за вычисления того как объекты сцены будут выглядеть в браузере на основе камеры и угла объекта. WebGLRenderer использует видеокарту для визуализации сцены:

```
var renderer = new THREE.WebGLRenderer();
```

Определение источника света:

```
var spotLight = new THREE.SpotLight( 0xffffff );
spotLight.position.set( -40, 60, -10 );
scene.add( spotLight );
```

Определение объекта(куб):

```
var cubeGeometry = new THREE.BoxGeometry(4,4,4);
var cubeMaterial = new THREE.MeshLambertMaterial({color: 0xff0000});
var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
scene.add(cube);
```

Вызов функции визуализации у объекта renderer:

```
renderer.render(scene, camera);
```

Работа с камерой:

Three.js имеет несколько элементов управления камерой, которые можно использовать для управления камерой в сцене.

Таб. . Элементы управления камерой

Имя	**Описание**
FirstPersonControls	Элементы управления FirstPersonControls ведут себя как в компьютерных играх от первого лица. Есть возможность перемещение с помощью клавиатуры и обзора вокруг с помощью мыши.
FlyControls	FlyControls - имитатора полета. Перемещение и управление с помощью клавиатуры и мыши.
RollControls	Более простая версия FlyControls. Позволяет передвигаться и катиться вдоль оси z.
TrackBallControls	Наиболее часто используемые элементы управления, которые позволяет использовать мышь (или трекбол) для легкого перемещения, панорамирования и масштабирования вокруг сцены.
OrbitControls	OrbitControls имитирует спутник на орбите вокруг определенной точки или объекта сцены. Перемещение с помощью мыши и клавиатуры.

При разработке инструментария пользователя и средств управления анатомическими объектами были использованы элементы управления FirstPersonControls, так как эти элементы реализуют интуитивно понятный интерфейс управления от первого лица с помощью мыши(обзор) и клавиатуры(перемещение) и обеспечивают лучшее впечатление от эффекта присутствия.

Таб. . Управляющие элементы FirstPersonControl

Управляющий элемент	**Действие**
Движение мыши	Осмотреться

← → ↑ ↓	Двигаться влево, вправо, вперед, назад
W	Двигаться вперед
A	Двигаться влево
S	Двигаться назад
D	Двигаться вправо
R	Двигаться вверх
F	Двигаться вниз
Q	Остановить управление

Constructor:

PerspectiveCamera(fov, aspect, near, far)

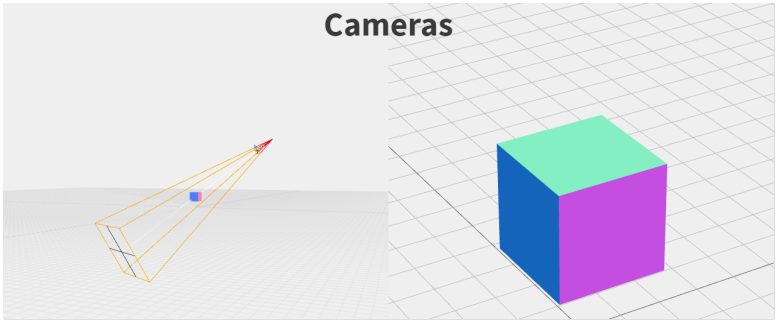
fov – вертикальный угол обзора

aspect – отношение между горизонтальным и вертикальным углом сцены

near – ближняя граница области визуализации.

far – дальняя граница области визуализации.

Рис. . Логика работы камеры



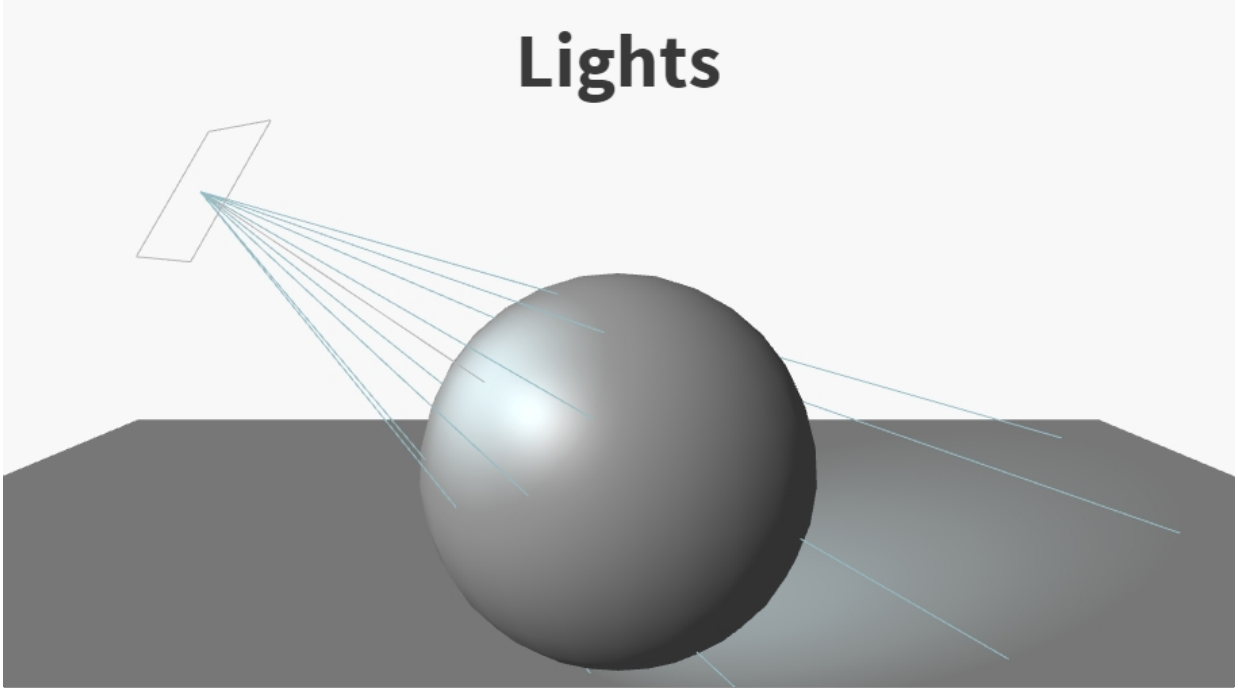
Настройка источников света:

Источники света являются важными элементами любой 3D-сцены. Без света мы не увидим ничего. Three.js содержит большое количество источников, каждый из которых имеет конкретное применение и собственное поведение:

Таб. . Источники света

THREE.AmbientLight	Основной свет, цвет которого добавляется к текущему цвету объектов в сцене
THREE.PointLight	Единственная точка в пространстве, из которого свет распространяется во всех направлениях. Этот свет не может быть использован для создания теней.
THREE.SpotLight	Источник света имеет конусообразную эффект, как у настольной лампы, пятна на потолке, или факела. Этот свет может отбрасывать тени.
THREE.DirectionalLight	Также называется бесконечный свет. Световые лучи от этого света можно рассматривать как параллельные, как, например, свет от солнца. Этот свет также может быть использован для создания теней.
THREE.HemisphereLight	Особый свет, который может быть использован для создания более естественного вида как открытом воздухе, имитируя освещение отражающей поверхности и слабо освещающего неба. Этот свет также не дает каких-либо функциональных возможностей для теней.
THREE.AreaLight	С помощью этого источника света, вместо одной точки в пространстве, вы можете указать район, из которого исходит свет. THREE.AreaLight не отбрасывает никаких теней.

Рис. . Логика работы источника света



В данной работе целью было применить источник света, который будет распространяться во всем пространстве сцены и не будет отбрасывать тени на другие объекты, в связи с этим, был использован наиболее практичный источник света THREE.AmbientLight. Также были добавлены два прожектора THREE.SpotLight, чтобы все объекты сцены были освещены с двух сторон.

Загрузка и сохранение анатомических объектов сцены

Three.js может читать несколько форматов 3D-файлов и импортировать геометрии и целые сцены, определенные в этих файлах. В следующей таблице приведены некоторые форматы файлов, которые поддерживаются Three.js:

Таб. . Форматы файлов моделей, поддерживаемых Three.js

JSON	Собственный формат Three.js. Несмотря на то, что это не официальный формат, он очень прост в использовании и обрабатывается очень удобно.
OBJ or MTL	OBJ является простым 3D-форматом, который впервые разработан Wavefront Technologies. Это один из наиболее широко распространенных форматов 3D-файлов и используется для определения геометрии объекта. MTL является форматом компаньоном OBJ. В файле MTL указан материал объектов, находящихся в файле OBJ.
Collada	Collada представляет собой формат для определения цифровых активов в формате XML. Это также широко используемый формат, который поддерживается почти всеми 3D приложениями и визуализаторами.
STL	STL - формат стереолитографии. Широко используется для быстрого прототипирования. Например, модели для 3D принтеров часто определяются в STL файлах.

Разработка геометрии каждой 3D-модели – трудоемкий процесс, которым занимаются профессиональные графические дизайнеры. В нашей работе не ставилась цель создания модели «с нуля». Поэтому с бесплатных источников были загружены готовые геометрии моделей анатомических объектов (кости скелета и человеческий мозг) в форматах Collada. При желании вы всегда можете преобразовать один формат в другой с помощью специальных программ--конвертеров.

Если вы посмотрите на исходный код программы, вы можете увидеть, что для некоторых из загружаемых моделей нам нужно изменить некоторые свойства материала или сделать масштабирование, чтобы модель оказалась

правильной. Причина, почему мы должны это делать, заключается в том, что модель создается в ее внешнем приложении (Blender, 3ds Max, 3ds Maya), придавая ей отличные размеры и группировку от той, что мы обычно используем в Three.js.

Пример загрузки модели мозга в формате Collada:

```
var loader = new THREE.ColladaLoader();
var brain;
loader.load("brain.dae", function (result) {

    var geometry = result.scene.children[0].children[0].geometry;  var
    mat = new THREE.MeshLambertMaterial({color: 0x7777ff, side:
    THREE.DoubleSide});

    brain = new THREE.Mesh(geometry, mat);
    brain.scale.set(3, 3, 3);
    scene.add(brain);

});
```

Средства управления анатомическими объектами

В данной главе описаны реализованные средства управления объектами и показаны программно-математическое обеспечение, методы и подходы, используемые при реализации средств управления.

Перемещение объектов с помощью мыши:

Изменение положения объектов сцены с помощью мыши было реализовано путем регистрации в программе определенных обработчиков события методами интерфейсов Web API:

document.addEventListener('mousedown', onDocumentMouseDown, false);

document.addEventListener('mousemove', onDocumentMouseMove, false);

document.addEventListener('mouseup', onDocumentMouseUp, false);

onDocumentMouseDown, onDocumentMouseMove, onDocumentMouseUp – объекты, которые принимают уведомление, когда событие указанного типа произошло. Это может быть объект, реализующий интерфейс EventListener или просто функция JavaScript. В представленной программе данные объекты – функции описанные в коде программы.

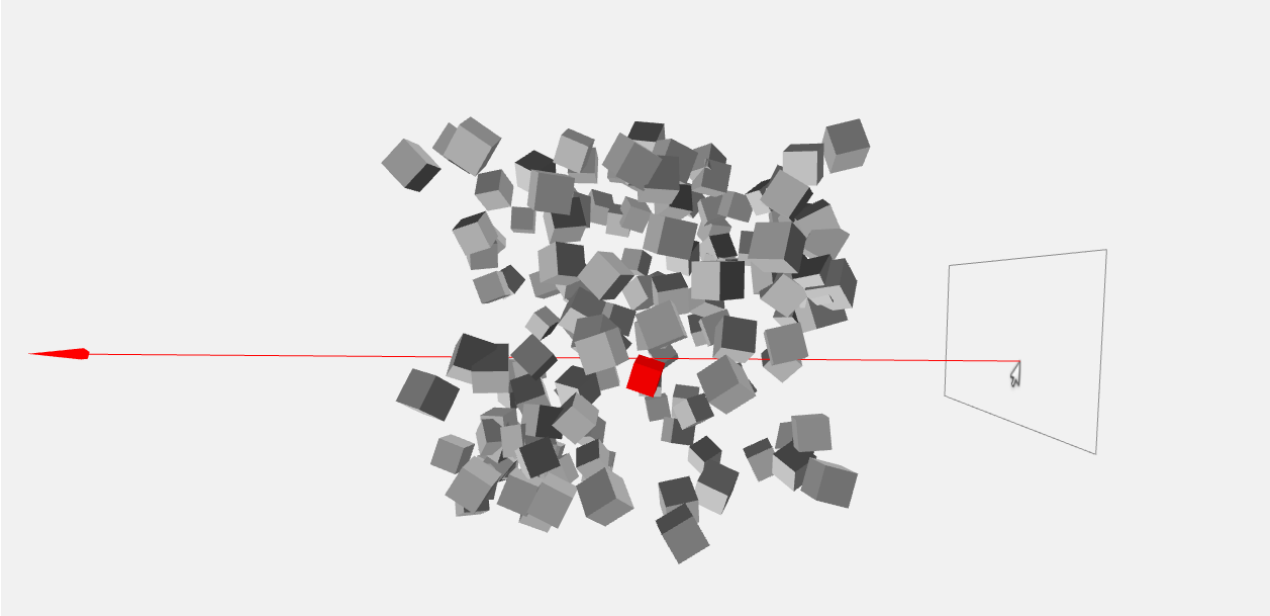
При нажатии левой кнопки мыши запускается onDocumentMouseDown. В коде всех функций мы используем THREE.Projector вместе с THREE.Raycaster, чтобы определить, нажали ли на конкретном объекте сцены. Что происходит, когда мы нажимаем на экране, выглядит следующим образом:

1. Во-первых, создается THREE.Vector3 на основе позиции, где мы щелкнули на экране.
2. Далее, с помощью функции vector.unproject, мы конвертируем щелкнутую позицию на экране в координаты в нашей Three.js сцене. Другими словами, мы проецируем координаты экрана в мировые координаты.

3. Далее, мы создаем THREE.Raycaster. С THREE.Raycaster, мы можем бросать лучи в нашей сцене. В этом случае мы испускаем луч от положения камеры (Camera.position) к позиции, которую мы щелкнули в сцене.

4. Используем функцию raycaster.intersectObjects, чтобы определить, какие из установленных объектов были поражены этим лучом.

Рис. . Иллюстрация работы THREE.Raycaster



5. Запоминаем первый пересеченный объект selection = intersects[0].object;

6. Сохраняем трехмерный вектор смещения точки на плоскости с которой пересекся луч относительно центра плоскости. Невидимая плоскость, которая всегда находится в позиции объекта, на который в данный момент наведен курсор и параллельна экрану. В этой плоскости и перемещается объект при данной позиции наблюдателя. offset.copy(intersects[0].point).sub(plane.position);

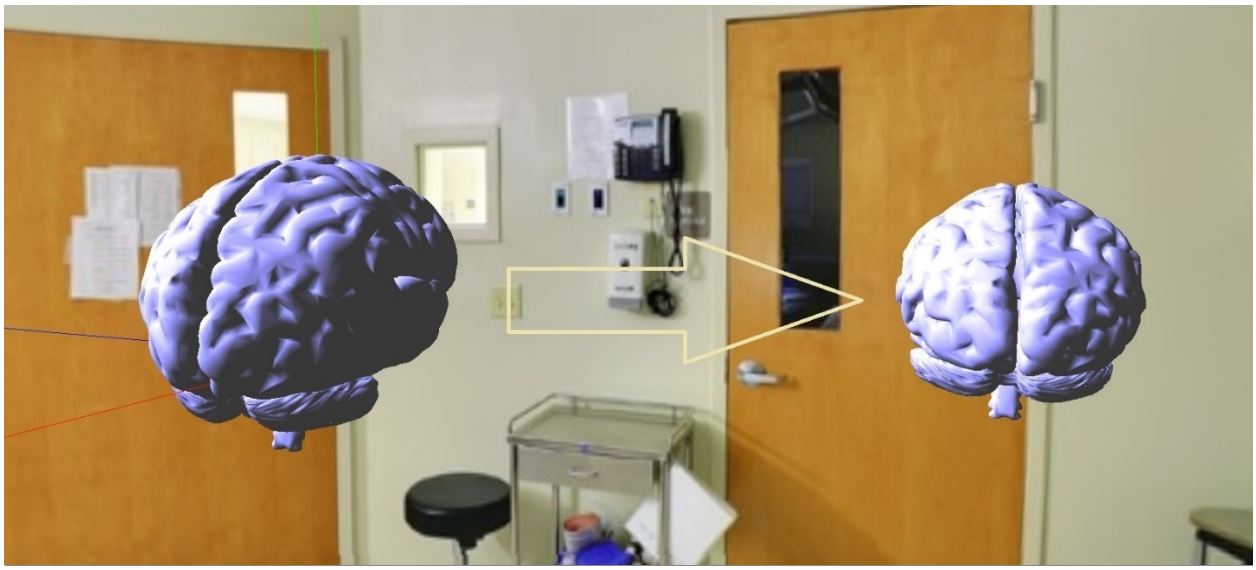
При перемещении курсора мыши запускается onDocumentMouseMove. Что происходит, когда мы перемещаем курсор на экране, выглядит следующим образом:

1. Во-первых, создается THREE.Vector3 на основе позиции, где курсор находится на экране.
2. Далее, с помощью функции vector.unproject, мы конвертируем позицию на экране в координаты в нашей Three.js сцене. Другими словами, мы проецируем координаты экрана в мировые координаты.
3. Далее, мы создаем THREE.Raycaster. С THREE.Raycaster, мы можем бросать лучи в нашей сцене. В этом случае мы испускаем луч от положения камеры (Camera.position) к позиции, которую мы щелкнули в сцене.
4. Если объект selection не пустой, смещаем позицию выбранного объекта (selection) на точку пересечения луча с плоскостью минус смещение offset, полученное при нажатии клавиши
5. Если объект selection пустой – перемещаем невидимую плоскость в позицию объекта на который наведен курсор.

При отпускании левой кнопки мыши запускается onDocumentMouseUp. Если клавиша отпускается, значит, перемещение объекта закончилось, поэтому необходимо обнулить текущий выбранный объект:

1. camControls.enabled = true;
2. selection = null;

Рис. . Перемещение объектов с помощью мыши в программе



Визуализация участков мозга.

Головной мозг является главным регулятором всех функций живого организма. Он представляет собой один из элементов центральной нервной системы. Строение и функции головного мозга — предмет изучения медиков до сих пор.

Строение головного мозга можно рассматривать в нескольких аспектах. Так в нем выделяют 5 главных отделов мозга: конечный (80% общей массы); промежуточный; задний (мозжечок и мост); средний; продолговатый. Также головной мозг разделяют на 3 части: большие полушария; ствол мозга; мозжечок.

В программе реализована визуализация следующих частей головного мозга:

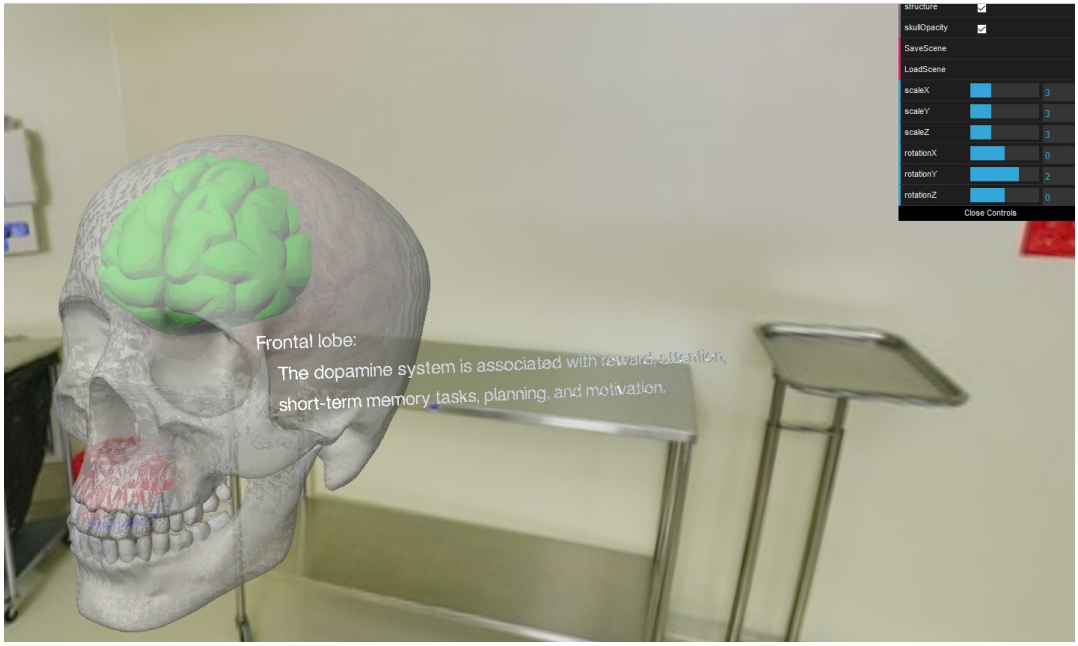
- Височная правая и левая доли
- Затылочная доля
- Теменная доля
- Лобная доля
- Мозжечок

- Гипофиз

Процесс визуализации:

При свободном полете камеры от первого лица, при наведении курсора мыши на определенную часть мозга, эта часть окрашивается в зеленый цвет, остальная часть мозга становится прозрачной. Также внутри сцены в текстовом виде выводится информация, содержащая описание выделенной части головного мозга. Этот текст помещается на расстояние равном половине расстояния между выделенной частью и камерой в данный момент и всегда повернут в сторону камеры, таким образом, описание участка мозга видно пользователю с любой позиции.

Рис. . Визуализация участков мозга в программе



Управление пространственными характеристиками объектов

Несколько сотрудников из Google создали библиотеку под названием dat.GUI, которая позволяет очень легко создавать простые и понятные компоненты пользовательского интерфейса.

Окно dat.gui в разработанном программном обеспечении выглядит следующим образом:

Рис. . Окно управления масштабом и поворота объекта

structure – если активна, включен режим визуализации участков мозга.

skullOpacity – если активна, череп становится прозрачным.

SaveScene – сохранение текущего состояния сцены (позиции и пространственные характеристики всех анатомических объектов).

LoadScene – восстановление сохраненного состояния сцены (все объекты получают сохраненные позиции и пространственные характеристики в сцене).

ScaleX, ScaleY, ScaleZ – три шкалы, изменяющие масштаб текущего объекта по X, по Y и по Z соответственно. Текущим объектом является последний “кликнутый” объект в сцене.

rotationX, rotationY, rotationZ – три шкалы, изменяющие поворот текущего объекта относительно точки его положения по X, по Y и по Z соответственно. Текущим объектом является последний “кликнутый” объект в сцене.

Close Controls – сворачивает окно dat.gui

Рис. . Исходное состояние анатомического объекта

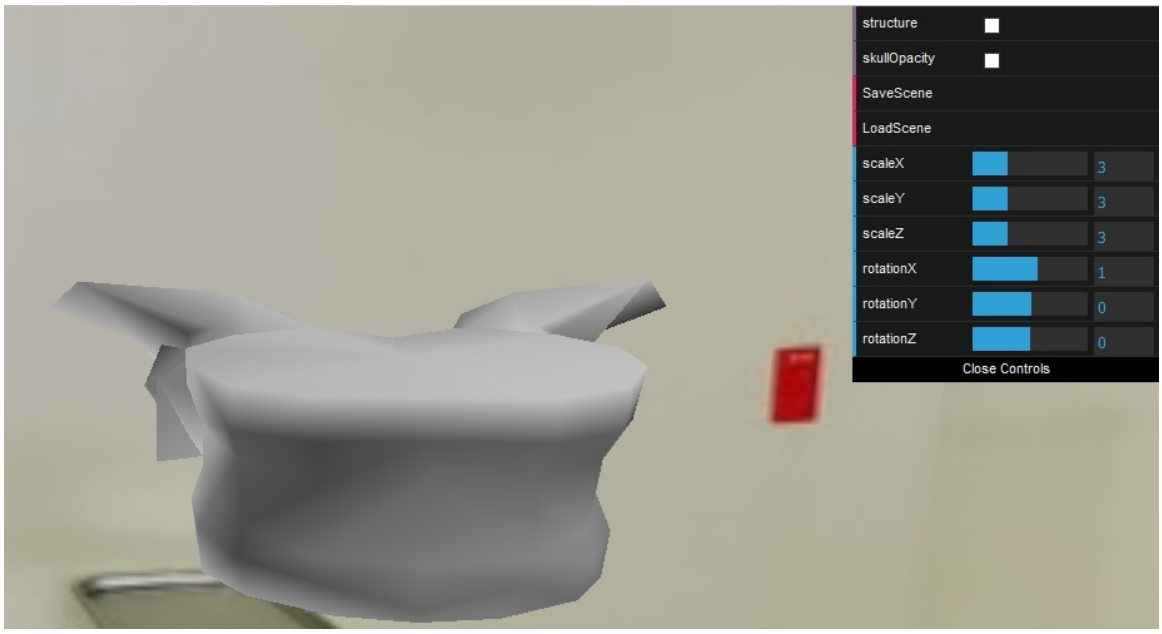


Рис. . Анатомический объект с измененным масштабом по X, Y и Z

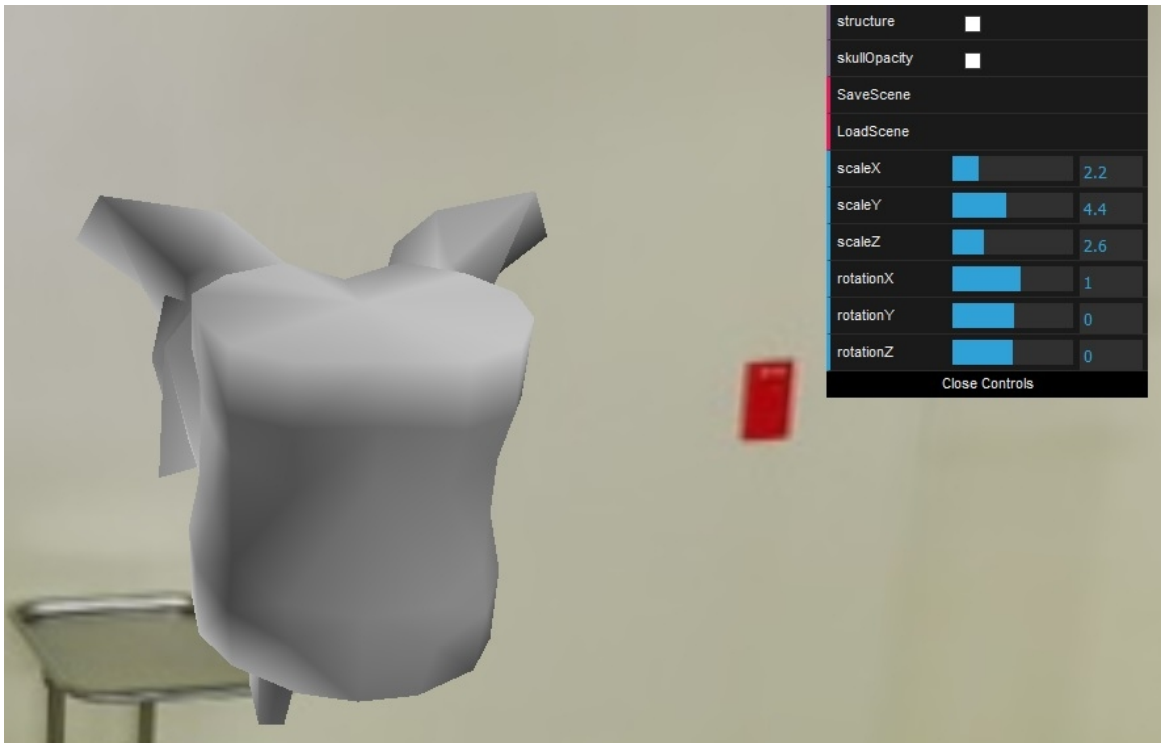
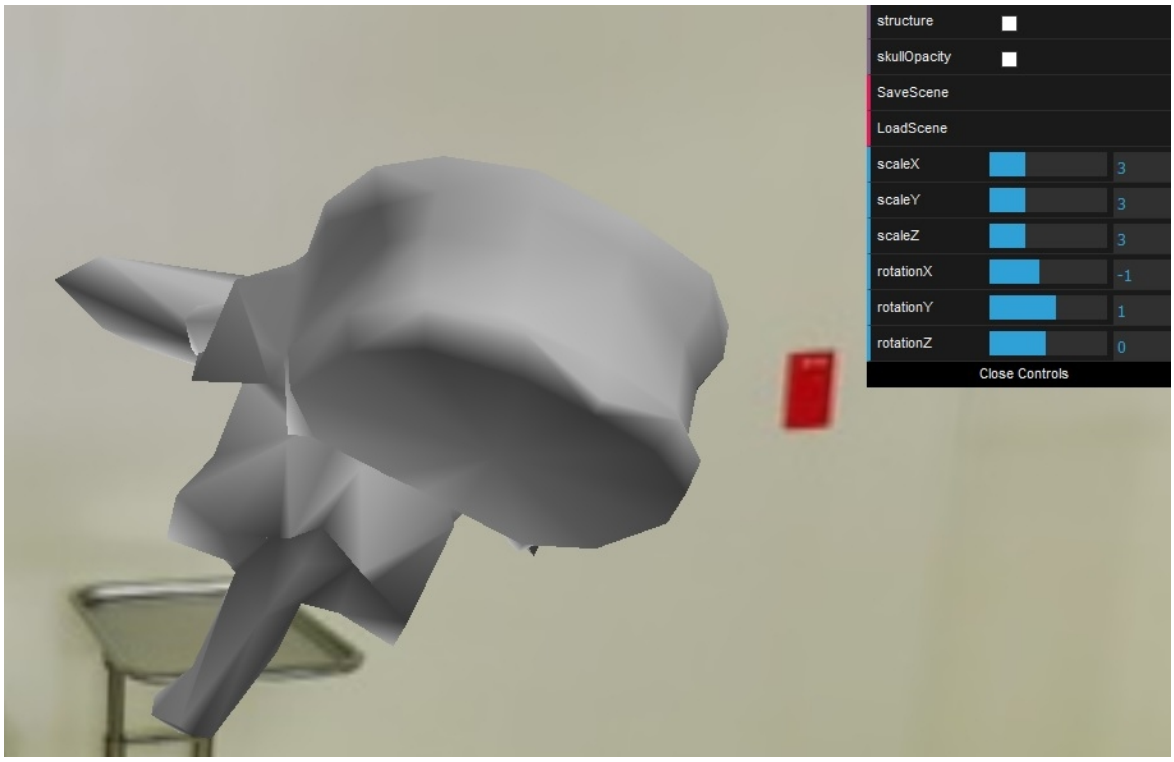


Рис. . Анатомический объект повернутый относительно X, Y и Z



Очки виртуальной реальности

Шлем виртуальной реальности (англ. *Head-mounted display*) — устройство, позволяющее частично погрузиться в мир виртуальной реальности, создающее зрительный и акустический эффект присутствия в заданном управляющим устройством (компьютером) пространстве. Представляет собой конструкцию, надеваемую на голову, снабженную видеоэкраном и акустической системой. Название «шлем» достаточно условное: современные модели гораздо больше похожи на очки, чем на шлем.

Шлем создаёт объёмное изображение, демонстрируя два изображения — по одному для каждого глаза. Кроме того, шлем может содержать гироскопический или инфракрасный датчик положения головы (трасер).

На рынке уже есть несколько производителей доступных моделей видеоочков: Oculus Rift, Sony HMZ, Gear VR, HTC Vive, Sony Project Morpheus и другие.

Типичные шлемы виртуальной реальности (очки виртуальной реальности) используют один или два дисплея с линзами и, иногда, зеркалами. В качестве дисплеев могут использоваться миниатюрные электронно-лучевые приборы, ЖК-дисплеи, LCos-проекторы, органические светодиоды. Иногда могут использоваться несколько микродисплеев для увеличения поля зрения.

Шлемы применяются военными и государственными службами, а также в гражданской и коммерческой области, например, в медицине, видеоиграх, спорте и тому прочее. Применяются в хирургии для изучения томографических снимков (компьютерная томография, магнитно-резонансная томография).

Освоение очков виртуальной реальности и их использование для создания зрительного эффекта присутствия при взаимодействии с анатомическими объектами стало одной из целей списка наших задач.

В работе были использованы очки фирмы Sony, модель HMZ-T1:

Рис. . Очки виртуальной реальности Sony HMZ-T1

Технические характеристики Sony HMZ-T1:

Требования к питанию

Модель для Северной Америки: 120 В переменного тока, 60 Гц

Модели для Великобритании, Гонконга и Европы: 220 В – 240 В, 50/60 Гц

Модель для России: 220 В – 240 В переменного тока, 50/60 Гц

Модель для Тайваня: 110 В переменного тока, 60 Гц

Модель для Китая: 220 В – 240 В переменного тока, 50/60 Гц

Потребляемая мощность

15 Вт (во время работы)

0,35 Вт (в режиме ожидания)

1,6 Вт (в режиме ожидания при использовании сквозного прохождения сигналов HDMI)

Рабочая температура

От 5 °C до 35 °C

Рабочая влажность

От 25 % до 80 %

Размеры (Ш×В×Г, включая максимально выступающие части)

Основной блок с креплением на голову: Приблиз. 210 мм × 126 мм × 257 мм

Блок процессора: Приблиз. 180 мм × 36 мм × 168 мм

Масса

Основной блок с креплением на голову: Приблиз. 420 г (без соединительного шнура)

Блок процессора: Приблиз. 600 г (без сетевого шнура переменного тока)

Расстояние между зрачками

55 мм – 72 мм

Наушники

Частотная характеристика: 12 Гц – 20000 Гц

Прилагаемые аксессуары

Сетевой шнур переменного тока (1)

Кабель HDMI (1)

Светозащитная глазная накладка (по 1 для левого и правого глаза)

Клипса дужки оголовья (1)

Держатель соединительного шнура (1)

Налобный суппортер (2)

Крышка налобного суппортера (1)

Справочное руководство (данное руководство) (1)

Руководство пользователя (1)

В представленной работе были рассмотрены современные подходы и виртуальные технологии в 3D-визуализации и связанные с ними понятия.

Использование очков виртуальной реальности позволило частично погрузиться в мир виртуальной сцены, рассмотреть медицинские объекты с различных сторон и углов обзора, снаружи и изнутри на основе камеры от первого лица, а также создать зрительный и акустический эффект присутствия.

Было отмечено удобство использования такого подхода при рассмотрении объектов и их особенностей. Очки позволили задать больший угол обзора в настройках камеры, а не ограничиваться размерами экрана монитора. Это позволило видеть больше количество анатомических объектов в данный момент и увеличило эффект присутствия. Также можно отметить, что использование Sony HMZ-T1 не вызывает усталость глаз. Дело в том, что в очках ваши глаза не сфокусированы на одном расстоянии, а постоянно смотрят на разные предметы на разных расстояниях. Поэтому зрачок находится в режиме, в котором он находится в реальной жизни.

Рис. . Просмотр разработанной сцены в очках виртуальной реальности

Киберболезнь (cybersickness)

Киберболезнь или киберукачивание (англ. *cybersickness*) — особое функциональное расстройство, вызываемое просмотром стереокино.

Основные симптомы:

-напряжение глаз

-нарушение ориентации в пространстве

-тошнота и рвота

Причина болезни:

Причиной болезни является сенсорный конфликт между частями зрительного аппарата. Когда трехмерный объект как бы покидает плоскость экрана, при взгляде на него глазные яблоки поворачиваются вовнутрь. Однако хрусталики продолжают фокусировать свет, исходящий от экрана, чтобы изображение сохраняло свою четкость. Несмотря на то, что положение зрачков изменилось, кривизна хрусталика остается прежней. Такое состояние не нормально и может вызвать приступ тошноты.

В современности:

Многие люди при просмотре трехмерных фильмов также испытывают чрезмерное напряжение глаз, головокружение, тошноту и позывы к рвоте. Джуди Барретт из Организации военных научно-технических исследований Австралии назвала это состояние «киберукачиванием» (cybersickness).

В руководстве пользователя очков Sony HMZ-T1 обозначены меры предосторожности для охраны здоровья. В том числе в руководстве показано, что некоторые люди могут испытывать дискомфорт (например, чрезмерное напряжение зрения, усталость, приступы тошноты или укачивание) во время просмотра видеоизображений или игр. Sony рекомендует всем зрителям регулярно делать перерывы при просмотре видео или во время игр. Продолжительность и частота необходимых перерывов зависят от особенностей каждого человека. Вы должны сами решить, что вам лучше подходит. Если вы испытываете дискомфорт, необходимо прекратить просмотр видео или игру до тех пор, пока эти симптомы не исчезнут; при необходимости проконсультируйтесь с врачом.

В ходе представленной работы, после продолжительного времяпровождения в очках виртуальной реальности было отмечено небольшое чувство дискомфорта, а именно: некоторое время болела голова, но не сильно, смотреть на настоящие объекты было не совсем приятно, перемещение казалось слишком медленным. Однако, через несколько минут подобное состояние проходило. Сильное чувство дискомфорта было испытано при рассматривании некоторых 3D-сцен, изображающих какую-либо местность, например, город, в которых камера могла поворачивать по углу крена. При таких поворотах земля постепенно становится небом, а небо землей. В таких случаях сразу наступает головокружение и начинают болеть глаза.

Дополнительные особенности и программные средства сцены

Панорама

В виртуальной сцене используется сферическая панорама, она же виртуальная панорма, она же 3D-панорама — особая панорамная фотография, предназначенная для показа на компьютере.

Виртуальная панорама содержит изображение охватом в 360 градусов по обоим осям. Вы можете посмотреть на предмет со всех сторон. Реализация

программы позволяет при повороте камеры с любого ракурса видеть сферическую проекцию загружаемой фотографии. Таким образом создается ощущение, что пользователь присутствует в месте, где была сделана фотография.

Рис. . 360°-фотография, использованная в программе в качестве панорамы

Средства отладки console.log

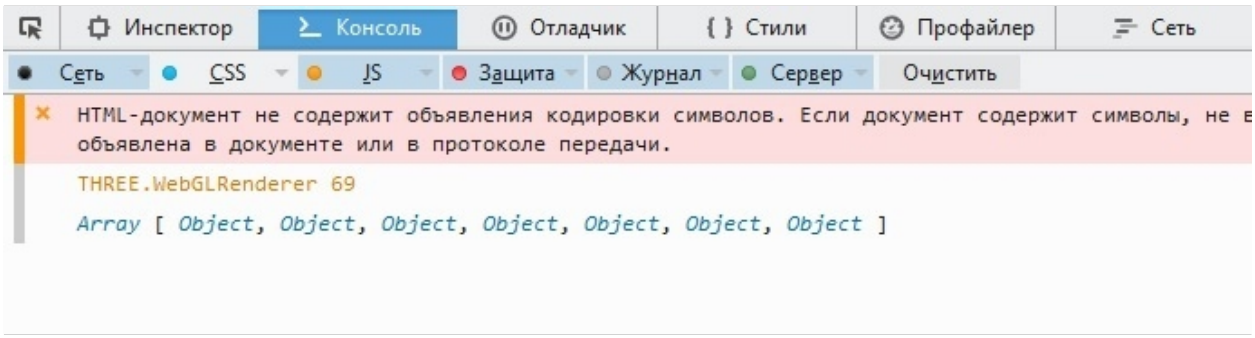
Console.log() - очень удобное средство для отладки, особенно, когда вы назвали свои объекты указав каждому свойство name. Это может быть очень полезным, в случае если нужно найти проблемы, связанные с конкретным объектом в вашей сцене.

Информация указанная в аргументе функции будет отображена в консоли браузера, который вы используете. Для того, чтобы открыть консоль

браузера, например, в Mozilla Firefox достаточно нажать Ctrl+Shift+K. В других браузерах необходимо зайти в Меню и найти пункт Инструменты разработчика.

Например console.log(scene.children); выдаст следующую информацию в консоль браузера:

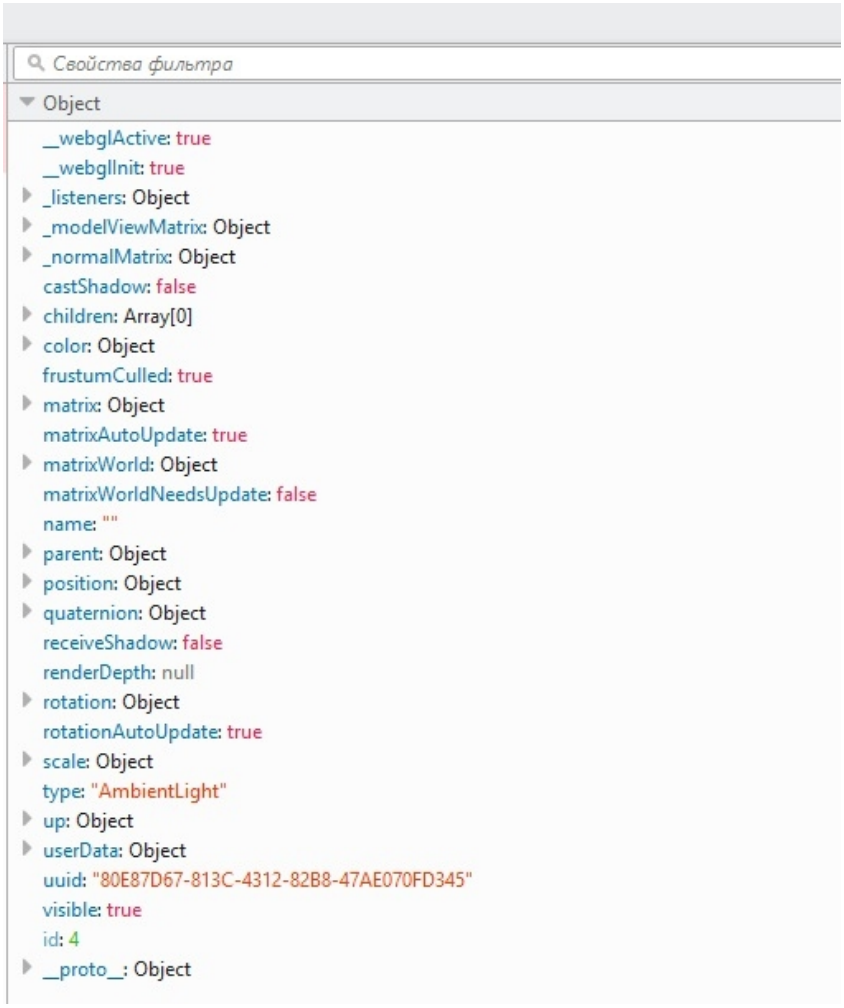
Рис. . Консоль браузера



Array [Object, Object, Object, Object, Object, Object, Object] - массив всех объектов нашей сцены, то есть детей объекта scene.

Кликнув мышью, например, по второму Object в списке, будет выведена следующая информация:

Рис. . Подробная информация о объекте в консоли браузера



В данном случае наш объект – THREE.AmbientLight, основной источник света, который был добавлен в сцену.

Локальное хранилище данных

Web Storage API предоставляет механизмы, при помощи которых браузеры могут безопасно хранить пары ключ/значение в интуитивно понятной манере.

В основе Веб хранилища лежат два механизма:

- хранилище сессии (sessionStorage) обслуживает область хранения данных для каждого домена, доступное на протяжении сессии. (пока браузер открыт, даже в случае перезагрузки страницы)
- локальное хранилище (localStorage) делает тоже самое, но сохраняет данные даже в случае, если переоткрыть браузер.

При написании программы для хранения промежуточных данных был использован localStorage. Данные из данного хранилища подгружаются только при вызове соответствующей функции, а не при каждой загрузке страницы. Подобным способом можно временно хранить и большие объёмы данных, при этом это не будет сказываться на производительности вашего сайта. Также данные, находящиеся в localStorage, хранятся бессрочно и будут сохранены даже после перезапуска работы компьютера. Однако при работе стоит помнить, что локальное хранилище у каждого браузера и каждого локального веб-сервера свое и данные, которые вы сохранили, например, тестируя свои примеры в Mangoose, не будут доступны из браузера Chrome, даже если локальный веб-сервер работает, используя этот браузер. Также стоит помнить, что размер localStorage не превышает 10 Мбайт.

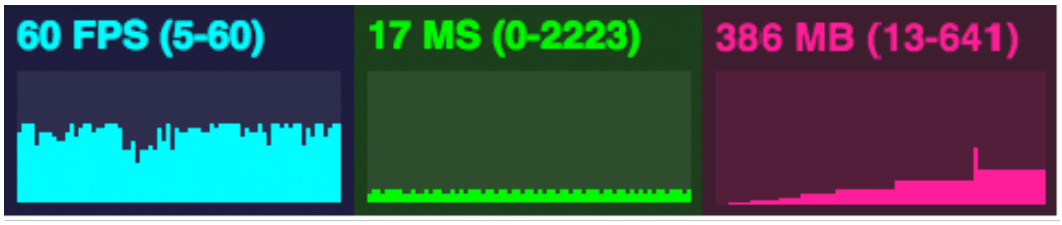
Окно stats.js

Если вы используете анимацию, можно включить в ваш код небольшую библиотеку, который дает нам информацию о частоте кадров в секунду. Эта библиотека, того же автора, что и Three.js, показывает небольшой график.

Этот класс обеспечивает простое информационное окно, которое поможет вам отслеживать эффективность кода:

- **FPS** Количество визуализированных кадров за последнюю секунду. Чем больше число, тем лучше.
- **MS** Количество миллисекунд, требующееся на визуализацию одного кадра. Чем меньше число, тем лучше
- **MB** Количество мегабайт выделенное в памяти.

Рис. . Виды информации в окне статистики



Программа

Полученное программное обеспечение предназначено для визуализации и управления трехмерными моделями различных объектов, в первую очередь связанных с медициной. Визуализация использует очки виртуальной реальности и работает через браузеры.

Программа написана на языке JavaScript, использует библиотеки API WebGL, в частности Three.js, и встроена в документ HTML. Формат исполняемого файла - «.html». Объем программы – 693 строк кода.

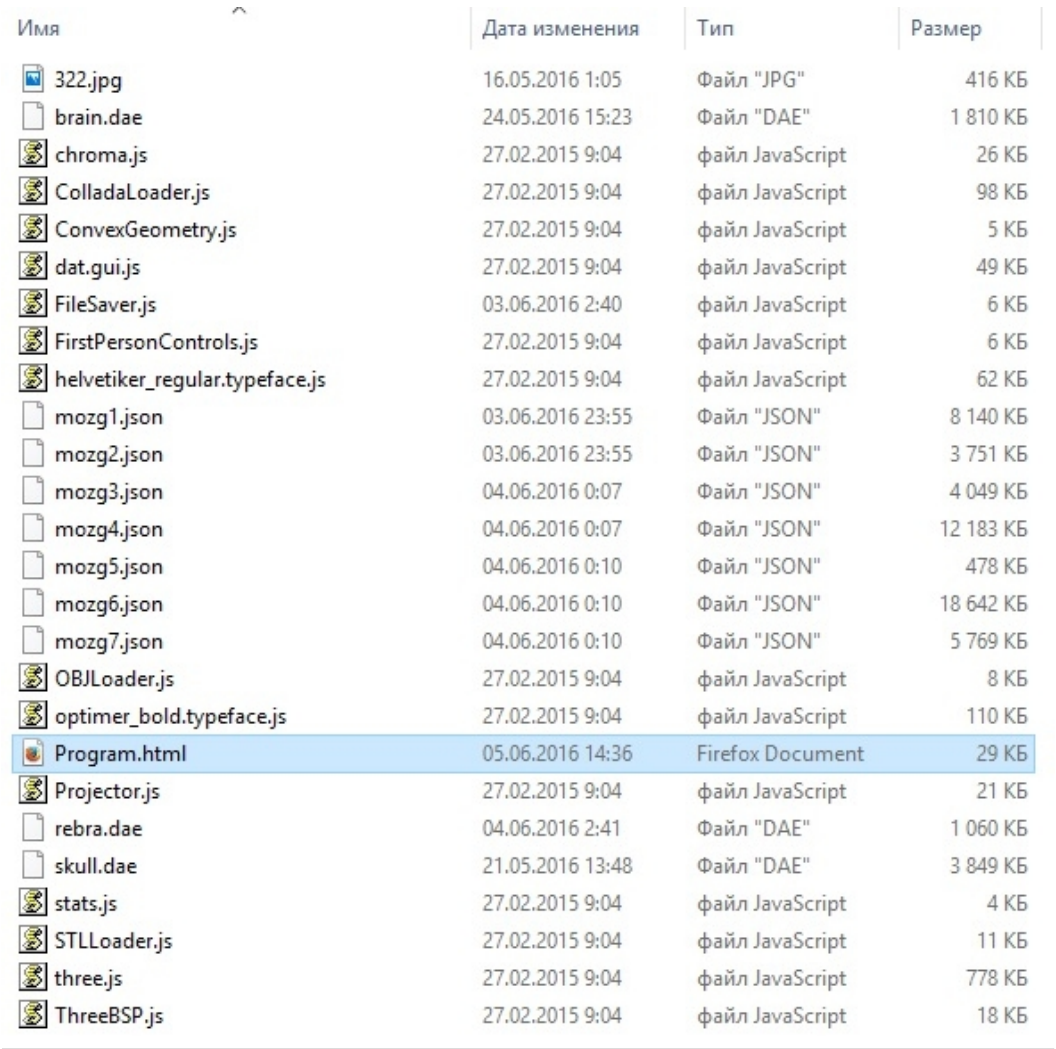
Для запуска программы необходимо настроить локальный веб-сервер, либо изменить настройки запуска вашего браузера, чтобы обеспечить доступ к подключаемым файлам.

В браузере программа требует около одного гигабайта оперативной памяти в активном режиме вкладки, в фоновом режиме объем используемой памяти может уменьшиться до пятисот мегабайт.

Для запуска программы необходимо поместить в папку с исполняемым файлом все подключаемые библиотеки и внешние файлы моделей и картинок.

Выходные данных отсутствуют. Чтобы проверить правильность работы программы запустите файл Program.html и проследите, что ваш браузер корректно открылся и в новой вкладке появилась сцена с первоначальным видом на панораму операционной комнаты. Управление от первого лица происходит с помощью мыши и клавиатуры (клавиши ← → ↑ ↓ и W, A, S, D). Клавиша Q – остановить полет камеры.

Рис. . Рабочая папка с программой



Имя	Дата изменения	Тип	Размер
322.jpg	16.05.2016 1:05	Файл "JPG"	416 КБ
brain.dae	24.05.2016 15:23	Файл "DAE"	1 810 КБ
chroma.js	27.02.2015 9:04	файл JavaScript	26 КБ
ColladaLoader.js	27.02.2015 9:04	файл JavaScript	98 КБ
ConvexGeometry.js	27.02.2015 9:04	файл JavaScript	5 КБ
dat.gui.js	27.02.2015 9:04	файл JavaScript	49 КБ
FileSaver.js	03.06.2016 2:40	файл JavaScript	6 КБ
FirstPersonControls.js	27.02.2015 9:04	файл JavaScript	6 КБ
helvetiker_regular.typeface.js	27.02.2015 9:04	файл JavaScript	62 КБ
mozg1.json	03.06.2016 23:55	Файл "JSON"	8 140 КБ
mozg2.json	03.06.2016 23:55	Файл "JSON"	3 751 КБ
mozg3.json	04.06.2016 0:07	Файл "JSON"	4 049 КБ
mozg4.json	04.06.2016 0:07	Файл "JSON"	12 183 КБ
mozg5.json	04.06.2016 0:10	Файл "JSON"	478 КБ
mozg6.json	04.06.2016 0:10	Файл "JSON"	18 642 КБ
mozg7.json	04.06.2016 0:10	Файл "JSON"	5 769 КБ
OBJLoader.js	27.02.2015 9:04	файл JavaScript	8 КБ
optimer_bold.typeface.js	27.02.2015 9:04	файл JavaScript	110 КБ
Program.html	05.06.2016 14:36	Firefox Document	29 КБ
Projector.js	27.02.2015 9:04	файл JavaScript	21 КБ
rebra.dae	04.06.2016 2:41	Файл "DAE"	1 060 КБ
skull.dae	21.05.2016 13:48	Файл "DAE"	3 849 КБ
stats.js	27.02.2015 9:04	файл JavaScript	4 КБ
STLLoader.js	27.02.2015 9:04	файл JavaScript	11 КБ
three.js	27.02.2015 9:04	файл JavaScript	778 КБ
ThreeBSP.js	27.02.2015 9:04	файл JavaScript	18 КБ

Заключение

В результате проделанной работы: реализован необходимый инструментарий и средства манипуляции анатомическими объектами, загруженными в сцену из внешних источников.

Реализована возможность просмотра сцены в очках виртуальной реальности в окне браузера.

Создан удобный пользовательский интерфейс для работы с объектами.

В ходе исследовательской работы были изучены современные средства и технологии в компьютерной графике, теории визуализации и человеко-компьютерного взаимодействия, была реализована виртуальная трехмерная сцена и возможность использования очков виртуальной реальности.

Левчуком Георгием была реализована возможность корректной загрузки различных форматов объектов в сцену, обеспечено управление пространственными характеристиками объектов, а также возможность встраивания 3D-сцены в html-документ и обеспечение корректной работы в современных браузерах.

В перспективе предполагается, что анализ моделей будет проводиться либо медиком, ведущим исследование и/или лечение больного, либо специалистом, проводящим 3D-принтинг для физической визуализации и иных объектов организма. Также работа может быть интересна всем, работающим в области компьютерной графики и визуализации данных, а также обычным пользователям виртуальных сред.

ЛИТЕРАТУРА

1. Jos Dirksen. Learning Three.js – the JavaScript 3D Library for WebGL. Second Edition., PACKT Publishing, 2015, 401 c.
2. Дацюк С. Ноу-хау виртуальных технологий - PC Club, №30, 1997.
3. Коичи Мацуда, Роджер Ли. WebGL Programming Guide: Interactive 3D Graphics: Programming with WebGL., 2015, 494 c.
4. Тихомиров Ю. Программирование трехмерной графики. – СПб.: BHV–Санкт-Петербург. 2000, 256 с.
5. Диего Кантор, Брэндон Джонс. Webgl beginner's guide. Kindle Edition. 2012. 376 с.
6. CadNav/3DModels/Hospital&Medical/Anatomy. [электронный ресурс]. Режим доступа: http://www.cadnav.com/3d-models/model-23877.html (дата обращения: 10.05.2016).

ПРИЛОЖЕНИЕ

```
<!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 {
         /* set margin to 0 and overflow to hidden, to go fullscreen */
         margin: 0;
         overflow: hidden;
      }
   </style>
</head>
<body>

<div id="Stats-output">
</div>
<!-- Div which will hold the Output -->
<div id="WebGL-output">
</div>

<!-- Javascript code -->
<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);
            }

    });

////////////////НАСТРОЙКА ВЫВОДА ТЕКСТА О ЧАСТИ МОЗГА//////////////////////////
```

```
        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));



                ////////////////////////////////////////////////////////////////////////////////
    // add the output of the renderer to the html element
    document.getElementById("WebGL-output").appendChild(webGLRenderer.domElement);
                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;



    // call the render function
    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);
        //webGLRenderer.clear();
        // render using requestAnimationFrame
        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) {
                    //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());
                    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>
```