МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Ярославский государственный университет им. П.Г. Демидова»

Кафедра вычислительных и программных систем

Сдано на кафедру
«____» _____ 2019 г.
Зав. кафедрой, к.т.н., доц.
_____ В. В. Васильчиков

КУРСОВАЯ РАБОТА

Веб-приложение на основе технологии WebGL для демонстрации модели солнечной системы.

по направлению подготовки 02.03.02 Фундаментальная информатика и информационные технологии

Научный руководитель
к. т. н., доц.
_____В.В. Васильчиков
«___» _____ 2019 г.

Студент группы ИТ-31
_____В.А. Новак
«___» _____ 2019 г.

Ярославль 2019 г.

Оглавление

Введение

Современные веб-браузеры позволяют отображать интерактивную трехмерную графику без использования сторонних плагинов или приложений (например, используя AdobeFlash). Разработка таких приложений происходит при помощи технологии WebGL. Эта технология использует широко распространённый язык JavaScript.

В отличие от настольных приложений работающих с 2D- и 3D-графикой, где чаще всего разработка осуществляется под определённую платформу, веб-приложения ограничиваются лишь поддержкой браузером технологии WebGL. Веб-приложения, построенные с использованием данной платформы, будут доступны в любой точке земного шара при наличии сети интернет вне зависимости от используемой платформы: то ли это десктопы с ОС Windows, Linux, Mac, то ли это смартфоны и планшеты, то ли это игровые консоли.

В данной курсовой работе на основе технологии WebGL разработано веб-приложение для демонстрации модели солнечной системы. Эта модель позволяет человеку составить более подробное представление о системе, в отличие, например, от книг, в которых изображения являются неподвижными и могут быть показаны только в 2D. Интерактивность модели заключается в том, что пользователь может при помощи мыши менять угол обзора, отдалять и приближать камеру.

1. Цель работы и постановка задач

Целью курсовой работы является создание интерактивного веб-приложения на основе технологии WebGL для демонстрации модели солнечной системы.

Поставленная цель определила постановку следующих задач:

- Изучение технологии WebGL;
- Разработка интерактивной модели солнечной системы;
- Формирование графического пользовательского интерфейса;
- Реализация возможности просмотра объектов солнечной системы с близкого расстояния.

Актуальность выбранной темы курсовой работы, как и актуальность разработки веб-приложений вообще, можно объяснить нижеследующими факторами:

- кроссбраузерность и отсутствие привязки к определенной платформе. Windows, MacOS, Linux - все это не важно, главное, чтобы ваш браузер поддерживал WebGL;
- использование языка JavaScript, который достаточно распространен;
- автоматическое управление памятью. В отличие от OpenGL в WebGL не надо выполнять специальные действия для выделения и очистки памяти;
- для технологии WebGL характерна высокая производительность, которая сравнима с производительностью нативных приложений.

2. Анализ предметной области

2.1. Назначение приложения

В тех предметах, где изучается солнечная система, чаще всего для её демонстрации используются двумерные картинки, которые не дают человеку чёткого представления о расположении планет относительно друг друга, о вращении планет, о их движении по орбитам и многом другом. Однако в связи с высоким развитием технологий в наше время существует возможность создавать трёхмерные модели солнечной системы, которые позволяют нивелировать недостатки 2D-картинок. Именно поэтому в рамках данной курсовой работы было разработано веб-приложение для демонстрации модели солнечной системы. Более того, модель является интерактивной, что ещё больше вызывает интерес к изучению и соответственно усвоению материала по изучаемому предмету.

2.2. Архитектура приложения

Перед началом разработки приложения необходимо было продумать его архитектуру и подобрать соответствующие технологии для её реализации. На рисунке 1 представлена схема архитектуры приложения.

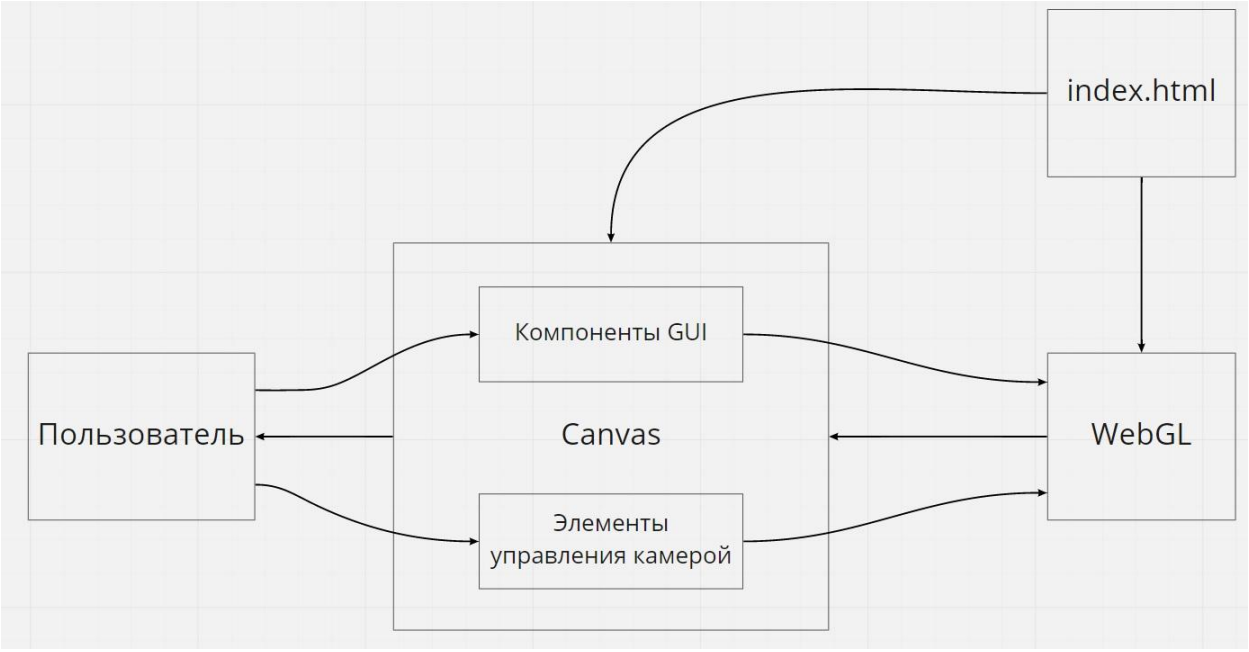

Рис. 1. Архитектура приложения.

Согласно схеме центральным и соответственно самым важным элементом является Canvas – элемент языка HTML5, предназначенный для создания растрового двухмерного изображения при помощи скриптов на языке JavaScript. Через него осуществляется процесс взаимодействия с пользователем и здесь же отображается результат взаимодействия.

На Canvas расположены компоненты графического пользовательского интерфейса и элементы управления камерой (поворот камеры, её отдаление и приближение), после взаимодействия с которыми полученные данные поступают на обработку программному коду WebGL. Обработав данные, WebGL возвращает набор пикселей, которые отображаются на Canvas и тем самым пользователь видит изменённую картинку у себя на экране.

Также в качестве “каркаса” здесь выступает html-документ index.html. Он необходим здесь лишь для инициализации и размещения тега canvas, для передачи контекста тега canvas (для работы с WebGL), а также для подключения скриптов.

2.3. Инструменты разработки

Рассмотрим основные инструменты для разработки целевого веб-приложения:

- HTML5 - язык для структурирования и представления содержимого всемирной сети интернет. Без него невозможно представить веб-разработку. Здесь используется 5-я версия языка, так как именно в ней был добавлен элемент Canvas, о назначении которого было рассказано в предыдушей подглаве;
- CSS – формальный язык, назначением которого является описание вида документа, написанного с использованием языка HTML5;
- JavaScript - прототипно – ориентированный сценарный язык программирования. JavaScript обычно используется, как встраиваемый язык для программного доступа к объектам приложений. Для программирования WebGL можно использовать любые языки

программирования, которые умеют работать с DOM API. JavaScript отлично для этого подходит за счёт простоты поддержки программного кода и широкой поддержки браузерами;

- WebGL - технология, базирующуюся на OpenGL ES и предназначенная для рисования и отображения интерактивной 2D- и 3D-графики в веб-браузерах. Для программирования на WebGL существует множество различных фреймворков. Однако наиболее популярным и активно развивающимся на сегодняшний день является Three.js. Его выбор обусловлен также тем, что на официальном сайте разработчика имеется подробная и доступная документация и огромное количество рабочих примеров, которые делают эту библиотеку одним из лидеров среди аналогичных систем.

Также стоит отметить, что в качестве среды разработки использовалась Visual Studio Code, разработанная компанией Microsoft. Приведём основные преимущества выбранной среды разработки:

- Имеются встроенные отладчик и командная строка;
- Поддержка практически всех языков программирования;
- Наличие встроенной библиотеки элементов кода;
- Наличие встроенных средств для тестирования, сборки, упаковки и развертывания приложений;
- Широкий набор настроек и кроссплатформенность.

2.4. Трехмерная компьютерная графика в веб-браузере

Первоначальные попытки по внедрению трехмерного контента в веб-браузер начались в 1994 году. На основе используемого языка HTML для представления веб-страниц необходимо было создать язык разметки для отображения трехмерного контента. Требованиями для нового языка были независимость от платформы, расширяемость и удобство использования даже при низкой пропускной способности Интернета. Консорциум Web3D, созданный для этой цели, разработал язык моделирования виртуальной

реальности (VRML). 26 мая 1995 года была опубликована окончательная спецификация VRML 1.0. В ней определяется простой текстовый формат, в котором объекты могут быть вложенными. Этими объектами могут быть камеры, источники света, материалы, а также представленные трехмерные объекты или преобразования. Для сохранения сцен указывается расширение файла .wrl. В 1997 году был завершен выпуск VRML 2.0 (VRML97), который был определен как стандарт ISO 14772-1: 1997. Помимо прочего, стандарт VRML 1.0 был дополнен анимацией и опциями взаимодействия с пользователем.

Существует несколько плагинов для браузера для отображения контента VRML, но практически ни один производитель браузеров не интегрировал стандарт непосредственно в свой продукт. В последующие годы было создано много других форматов файлов для хранения 3D-сцен с акцентом на отображение их в браузере. Помимо прочего, формат X3D на основе XML был унаследован от VRML. Консорциум Web3D предлагает полностью интегрировать стандарт X3D в HTML5, но W3C пока отказывается: «Встраивание трехмерных изображений в документы XHTML является областью X3D или технологий, основанных на X3D, которые поддерживают пространство имен». Подобно SVG или MathML, должна быть возможна нативная презентация без плагина.

Представление двумерной и трехмерной графики в браузере в основном может осуществляться двумя различными способами: во-первых, можно объявить граф сцены как часть HTML-документа. Таким образом, отдельные объекты графики находятся наравне с элементами HTML на странице. Каждый объект графики интегрирован в дерево DOM и может быть выбран и изменен с помощью CSS или JavaScript. Для отображения двумерной графики формат SVG включен в стандарт HTML5. Эквивалент для трехмерной графики, X3D, не является частью стандарта. С другой стороны, графика может быть обязательно нарисована на поверхности рисования. Элемент рисования использует элемент canvas и является

единственным объектом графического объекта, привязанным к дереву DOM. Содержимое, нарисованное на артборде, не может быть изменено. Область рисования может быть освобождена только полностью или в разрезе и / или перенесена. HTML5 определяет 2D-возможности элемента canvas и использует WebGL для 3D-контента.

3. Технология WebGL

WebGL (Web-based Graphics Library) - программная библиотека для языка программирования JavaScript, позволяющая создавать на JavaScript интерактивную 3D-графику, функционирующую в широком спектре совместимых с ней веб-браузеров. За счёт использования низкоуровневых средств поддержки OpenGL, часть кода на WebGL может выполняться непосредственно на видеокартах. WebGL - это контекст элемента canvas HTML, который обеспечивает API 3D графику без использования плагинов. Спецификация версии 1.0 была выпущена 3 марта 2011 года. Проект по созданию библиотеки управляется некоммерческой организацией Khronos Group.

3.1. Описание графического конвейера

WebGL, как и любой современный графический интерфейс, отвечает за два главных параметра: однородные координаты и цвета. Оба параметра могут расчитываться и задаваться с помощью специальных программ для определенных ступеней графического конвейера, используемого в трехмерной графике для определения окончательных параметров объекта или изображения. Эти программы называются шейдерами и для их написания в OpenGL используется специальный шейдерный язык GLSL. Так как WebGL представляет собой некую обертку OpenGL ES спецификации, которая является подмножеством графического интерфейса OpenGL, в нём так же используется стандартный графический конвейер OpenGL, представленный в упрощенной версии на рисунке 2.

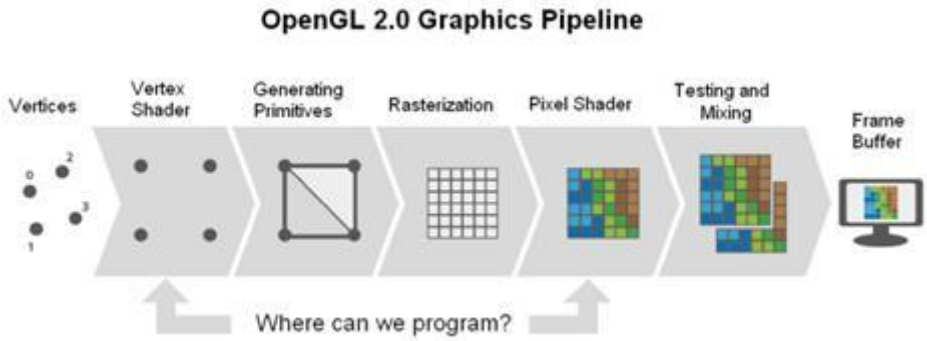

Рис. 2. OpenGL графический конвейер.

Конвейер представляет собой последовательность состояний, выполняющихся параллельно и в фиксированном порядке. Графический конвейер - это некоторое программно-аппаратное средство, которое преобразует объекты, описанные в виртуальном мире в матрицу ячеек видеопамяти растрового дисплея. Каждое состояние конвейера получает данные из предыдущего состояния, обрабатывает их и передает следующему. Процесс получения итогового изображения можно коротко описать следующим образом: программа заполняет управляемые OpenGL буферы памяти массивами из вершин, эти вершины проецируются в пространство экрана, преобразуются в треугольники и растеризуются в фрагменты размером с пиксель, которым присваиваются цветовые значения, и далее они отображаются в экранном буфере. Получив первое представление о совершаемых операциях внутри конвейера, рассмотрим каждый этап подробней.

Первоначально на вход поступает один или несколько вершинных буферов, наполненных массивами вершинных атрибутов. Набор этих буферов, поставляющих данные для работы рендеринга, в совокупности называется массивом вершин. Каждая вершина имеет определенный набор атрибутов, таких как позиция в 3D пространстве, цвет, текстурные координаты, которые связывают вершину с точкой на одной или нескольких текстурах, вектор нормали и другие. Эти атрибуты используются в качестве входных данных в вершинный шейдер. В последствие над ними производятся математические преобразования, в ходе которых может

происходить трансформация позиции вершины, генерация или преобразования текстурных координат, расчет освещения для каждой отдельной вершины с помощью нормали и т.д. Так же на вход вершинному шейдеру, помимо вершинных атрибутов могут подаваться единые (uniform) переменные с определенными значениями, которые могут использоваться для всех вершин сразу. На последнем шаге данного этапа, массив вершин снабжается дополнительным массивом индексов, который определяет, какие вершины попадут в конвейер.

На втором этапе осуществляется построение примитивов из трансформированных вершин с помощью информации об их соединениях, содержащейся в ранее упомянутом массиве индексов. Графический процессор соединяет спроецированные вершины в треугольники, используя порядок индексов, и группирует их в наборы из трех. Таким образом, осуществляется сборка геометрических примитивов, состоящих из последовательностей треугольников, линий и точек. Стоит отметить, что вершины могут быть сгруппированы разными способами:

- Каждые три элемента являются независимым треугольником.
- Элементы могут образовывать треугольную полоску, используя последние две вершины каждого треугольника как первые две следующего.
- Элементы могут образовывать «вентилятор» из треугольников, каждый из которых будет содержать общую вершину.

На рисунке 3 графически представлены описанные способы группировки.

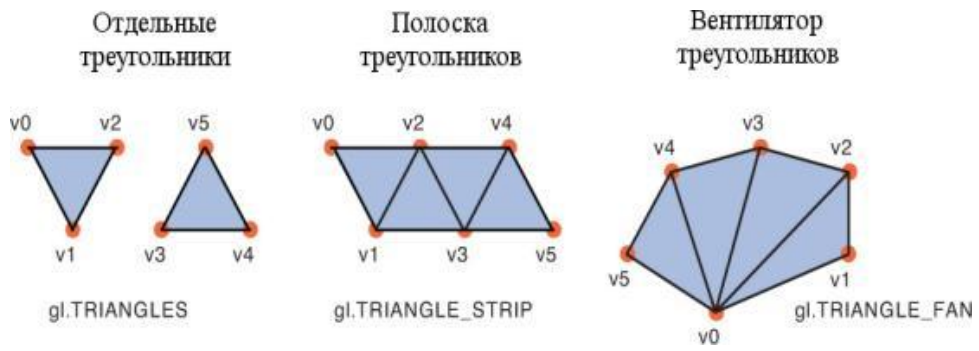

Рис. 3. Типы соединений треугольников.

После построения примитивов наступает третий этап - растеризация. После освобождения графического процессора от ненужных вычислений начинается работа растеризатора. Во время его функционирования все треугольники, составляющие объекты, прошедшие отсечение, преобразуются в массив пикселей. Каждый растеризуемый пиксель обладает теми же атрибутами, что и вершина после обработки в вершинном шейдере. Цветовые значения атрибутов пикселя вычисляются на основе линейной интерполяции вершин примитива. Если вершине было присвоено цветовое значение, то растеризатор произведет смешивание этих цветов вдоль поверхности, состоящей из пикселей (Рис. 4.). После данного этапа в конвейер передается изображение в виде фрагментов, представляющих растры треугольников.

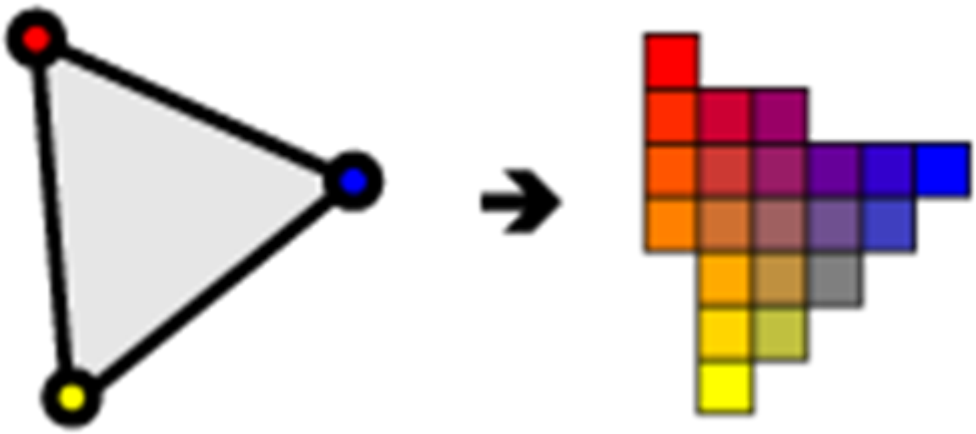

Рис. 4. Пример смешивания цветов.

Сгенерированные на предыдущем этапе фрагменты поступают в программу под названием пиксельный (фрагментный) шейдер, принимающей

в качестве входных параметров изменяемые (varying) переменные из вершинного шейдера, если они были инициализированы, а так же интерполированные значения, полученные в процессе растеризации. Основные операции, осуществляемые внутри фрагментного шейдера, включают в себя: отображение текстур, расчёт света и задание глубины. В качестве итога всех операций и математических преобразований мы получаем конечный цвет каждого фрагмента.

Перед тем как конечный результат конвейера поступит в кадровый буфер и отобразится на экране пользователя, проводится ряд пиксельных тестов над фрагментами, полученными в ходе растеризации и текстурирования. После прохождения данной процедуры отсеивания фрагменты становятся пикселями. Каждый фрагмент имеет соответствующую пикселю координату (x, y), глубину (z) и цвет (r, g, b, a). Каждый фрагмент последовательно проходит обработку каждым из следующих тестов:

- Тест отсечения (scissor test).
- Тест прозрачности (alpha test).
- Тест трафарета (stencil test).
- Тест глубины (depth test).
- Тест смешивания (blend test).

Эти тесты определяют конечный вид, цвет и глубину фрагмента. Если в каком-то тест происходит ошибка, то фрагмент удаляется и не попадает в следующий. Тест отсечения позволяет определить, лежит ли фрагмент в прямоугольнике видимости. Тест прозрачности позволяет принять или отклонить фрагмент на основе его альфа канала. Тест трафарета используется для ограничения области рисования некоторых участков экрана. Тест буфера глубины удаляет все невидимые фрагменты объектов, находящихся за уже нарисованными. Тест смешивания - это логическая операция, которая комбинирует компоненты цвета r, g, b и альфа канала входящего фрагмента с

аналогичными компонентами пикселя уже сохраненного в данном месте экрана. Последнее действие выполняется именно на этом этапе, т.к. фрагментный шейдер не имеет доступа к кадровому буферу. Конечный результат всех произведенных преобразований и операций отправляется на устройство видеовывода в виде растрового изображения.

3.2. Распространение и реализация в разных браузерах

В настоящее время WebGL поддерживается основными современными браузерами (Рис. 5.) как на настольных, так и на мобильных платформах (например, Google Chrome на Android или Internet Explorer на Windows Phone). Чтобы гарантировать большую плавность и стабильность, очень часто в среде Windows браузеры используют слой графической абстракции, называемый ANGLE (почти Native Graphics Layer Engine), предназначенный для преобразования вызовов OpenGL в вызовы Direct3D с использованием одного из двух доступных внутренних интерфейсов. (Direct3D 9.0c или Direct3D 11, в зависимости от того, насколько свежая видеокарта). В Chrome и Firefox по умолчанию используется ANGLE, но при этом пользователь может переключаться на собственное выполнение кода OpenGL.

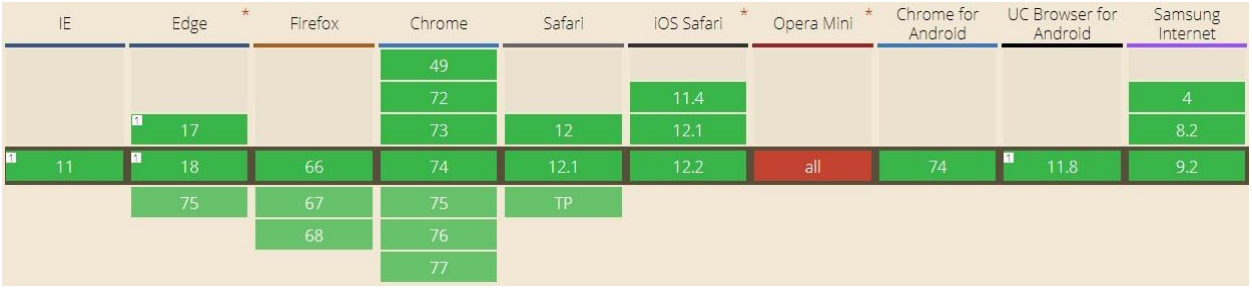

Рис. 5. Поддержка различными браузерами.

3.3. Преимущества использования WebGL

Преимущества и актуальность WebGL, как и актуальность разработки веб-приложений вообще, можно объяснить нижеследующими факторами:

- кроссбраузерность и отсутствие привязки к определенной платформе. Windows, MacOS, Linux - все это не важно, главное, чтобы браузер поддерживал WebGL;
- использование языка JavaScript, который достаточно распространен;
- автоматическое управление памятью. В отличие от OpenGL в WebGL не надо выполнять специальные действия для выделения и очистки памяти;
- поскольку WebGL для рендеринга графики использует графический процессор на видеокарте (GPU), то для этой технологии характерна высокая производительность, которая сравнима с производительностью нативных приложений;
- технология WebGL поддерживается большинством популярных браузеров;

3.4. Библиотека THREE.js

При разработке веб-приложения использовалась легковесная кроссбраузерная библиотека Three.js, написанная на языке программирования JavaScript. Автором этой библиотеки является Ricardo Cabello, он же Mr.doob, а так же большое количество сторонних разработчиков - контрибьютеров. Исходный код расположен в репозитории GitHub - крупнейшем веб-сервисе для хостинга IT - проектов и их совместной разработки, что позволяет каждому внести свой вклад в разработку. К ключевым особенностям данной библиотеки можно отнести:

- Рендереры: Canvas, SVG или WebGL;
- Сцена: добавления и удаления объектов в режиме реального времени;
- Камеры: перспективная или ортографическая;

- Анимация: каркасы, быстрая кинематика, обратная кинематика, покадровая анимация;
- Источники света: внешний, направленный, точечный; тени: брошенные и полученные;
- Шейдеры: полный доступ ко всем OpenGL шейдерам (GLSL);
- Объекты: сети, частицы, спрайты, линии, скелетная анимация и другое;
- Геометрия: плоскость, куб, сфера, тор, 3D текст и другое; модификаторы: ткань, выдавливание
- Загрузчики данных: двоичный, изображения, JSON и сцена;
- Экспорт и импорт: утилиты, создающие Three.js-совместимые JSON файлы из форматов:Blender, openCTM, FBX, 3D Studio Max, и Wavefront.obj файл;
- Поддержка: документация по API библиотеки находится в процессе постоянного расширения и дополнения, есть публичный форум и обширное сообщество;
- Примеры: на официальном сайте можно найти более 150 примеров работы со шрифтами, моделями, текстурами, звуком и другими элементами сцены.

Использование библиотеки позволяет значительным образом сэкономить время, необходимое для создания тестового прототипа, т.к. большинство базовых элементов, используемых при 3D визуализации, уже разработано и нет необходимости тратить время на изучение множества методов и подходов по реализации тех или иных техник и алгоритмов, а так же их оптимизации.

Опишем предоставляемые библиотекой объекты, играющие наиболее важную роль при работе с 3D пространством:

- THREE.Scene - пожалуй главный элемент, необходимый для создания иерархии объектов. Сцена содержит все созданные и

добавленные в неё графические объекты. По своей сути представляет граф, содержащий общую структуру данных, используемую для хранения и логической организации его элементов. В THREE библиотеке все объекты существуют в связке родитель - ребенок. Таким образом, у каждого элемента графа или дерева может быть много детей и только лишь один родитель. Данный способ хранения объектов 3D пространства удобен тем, что все эффекты и операции, применяемые на родителе, автоматически распространяются на всех его потомков. Хорошим примером подобной операции является перемножение геометрических матриц преобразования. Каждый объект дерева обладает подобной матрицей, которая задает его положение, вращение и размер внутри его родителя. Благодаря этому свойству, целые группы объектов, имеющие общего родителя, могут перемещаться и трансформироваться как одно целое.

- THREE.Object3D - представляет собой базовый класс - узловой элемент, используемый в графе сцены. Содержит все необходимые свойства и методы, используемые при работе с 3D объектами внутри дерева, такие как: уникальный индентификатор, родитель, массив детей, позиция, вращение, геометрическая матрица преобразования, необходимость отсечения и т.д.
- THREE.Camera - абстрактный класс, используемый для построения различных типов виртуальных камер для отображения заданной области виртуального мира. Среди реализованных имеется перспективная и ортогональная.
- THREE.Mesh - базовый класс предназначенный для создания различных объектов, содержащих геометрию и материал. Класс THREE.Geometry содержит всю необходимую информацию, необходимую для описания 3D модели. При создании класса,

содержащего логику построения произвольной модели, он должен быть унаследован. Базовый класс THREE.Material описывает внешний вид объектов. При создании разных типов материалов должен быть унаследован.

4. Реализация приложения

4.1. Создание HTML-документа

В интернете практически нет веб-приложения, которое не было бы связано как-либо с языком разметки гипертекста HTML. Так и разрабатываемое приложение не стало исключением. Ниже приведён код файла index.html:

```
<!DOCTYPE html>
<html>
      <head>
         <title>Солнечная система</title>
         <meta charset="UTF-8">
         <meta name="viewport" content="width=device-width">
         <link rel="stylesheet" type="text/css" href="styles.css"/>
      </head>
      <body>
         <div id="webgl"></div>
         <div id="description" class="info__panel"></div>
         <script src="js/dat.gui.min.js"></script>
         <script src="js/OrbitControls.js"></script>
         <script src="js/three.js"></script>
         <script src="main.js"></script>
      </body>
</html>
```

В данном коде особое внимание стоит уделить следующим элементам:

- `<link rel="stylesheet" type="text/css" href="styles.css"/>` - тег, устанавливающий связь с внешним файлом со стилями (styles.css)
- `<div id="webgl"></div>` - именно в этом теге происходит рендеринг всей сцены
- `<div id="description" class="info__panel"></div>` - вспомогательный контейнер, в котором при необходимости отображается информация об объекте солнечной системы
- скрипт `<script src="js/dat.gui.min.js"></script>`, подключающий JavaScript-библиотеку, которая позволяет создать простой графический

интерфейс пользователя. Исходный код библиотеки можно скачать с https://github.com/dataarts/dat.gui

- скрипт `<script src="js/OrbitControls.js"></script>`, подключающий JavaScript-библиотеку, которая позволяет добавить элементы управления для работы с камерой
- скрипт `<script src="js/three.js"></script>`, подключающий библиотеку Three.js. Исходный код Three.js и OrbitControls.js можно скачать с https://github.com/mrdoob/three.js/
- скрипт `<script src="main.js"></script>`, подключает файл main.js, где содержится весь код программы

Остальные теги такие, как `<!DOCTYPE html>`, `<html>`, `<head>`, `<body>`, `<title>`, `<meta>` являются достаточно обыденными для любого html-документа и не нуждаются в рассмотрении.

4.2. Инициализация основных объектов Three.js

Далее рассмотрим добавление основных элементов библиотеки Three.js. Так в файле main.js происходит единственный вызов функции init(), в которой инициализируются основные объекты Three.js, а также создаются объекты солнечной системы.

Код функции init():

```
function init() {
	setRenderer();
	setScene();
	setCamera();
	setControls();
	setSun();
	createVisibleOrbits();
	createPlanets();
	setGUI();
	setRaycaster();
	window.addEventListener( 'resize', onWindowResize, false );
	update(renderer, scene, camera, controls);
}
```

Коротко о назначении некоторых функций:

- `setRenderer()` - создаёт рендерер (WebGLRenderer), который отвечает за отображение сцены
- `setScene()` - создаёт сцену, на которую в дальнейшем будут добавлены объекты солнечной системы, и устанавливает фон сцены
- `setCamera()` - создаёт перспективную камеру, при помощи которой пользователь может видеть сцену
- `setControls()` - с помощью ранее подключённой библиотеки OrbitControls.js, создаёт элементы управления камерой такие, как приближение, отдаление и перемещение камеры
- `window.addEventListener` - обработчик события resize: при изменении размеров окна браузера изменяется соотношение сторон камеры и сторон контейнера с id=”webgl". Это нужно для правильного отображения сцены при уменьшении или увеличении окна браузера

Остальные функции будут разобраны в последующих подглавах. На рисунке 6 показано то, как выглядит сцена, если применить только вышеописанные функции.

Рис. 6. Сцена с фоном звёздного неба.

4.3. Добавление Солнца на сцену

Центром солнечной системы является Солнце. Поэтому рассмотрим функцию его добавления на сцену. Для этого используется функция `setSun()`. Так как планеты, спутники и Солнце имеют сферическую форму, то логично добавить функцию для создания сферы. Таковой является функция `getSphere()`, код которой представлен ниже:

```
1 function getSphere(material, size, segments) {
2     var geometry = new THREE.SphereGeometry(size, segments,
segments);
3     var obj = new THREE.Mesh(geometry, material);
4     obj.castShadow = true;
5     return obj;
6 }
```

На вход подаются 3 параметра: material - внешний вид (цвет, текстура и другие), size - радиус сферы, segments - количество треугольников, образующих полигональную сетку, в ширину и высоту. В строке 2 происходит инициализация сферы, а уже в строке 3 на неё накладывается текстура. В качестве результата возвращается текстурированная сфера.
Тогда за создание Солнца отвечает следующий код:

```
var sunMaterial = getMaterial("basic", "rgb(255, 255, 255)", new
THREE.ImageUtils.loadTexture("img/sunmap.jpg"));
sun = getSphere(sunMaterial, 16, 48);
sun.name = 'sun';
scene.add(sun);
```

Здесь функция `getMaterial()` по введённым параметрам возвращает нужный материал (внешний вид). Результат кода изображён на рисунке 7.

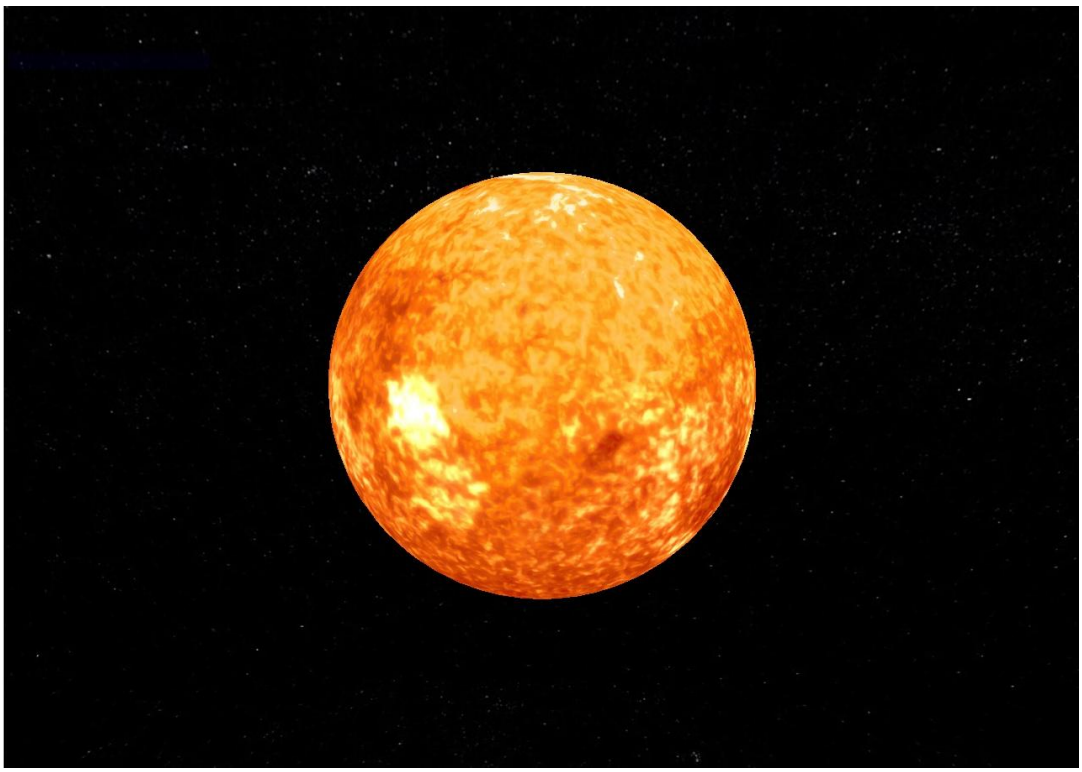

Рис. 7. Добавление Солнца на сцену.

Для имитации свечения Солнца используется спрайт:

```
var spriteMaterial = new THREE.SpriteMaterial({
      map: new THREE.ImageUtils.loadTexture("img/glow.png"),
      useScreenCoordinates: false,
      color: 0xffffee,
      transparent: false,
      blending: THREE.AdditiveBlending });
var sprite = new THREE.Sprite(spriteMaterial);
sprite.scale.set(48, 48, 1.0);
sun.add(sprite);
```

Результат применения спрайта изображен на рисунке 8.

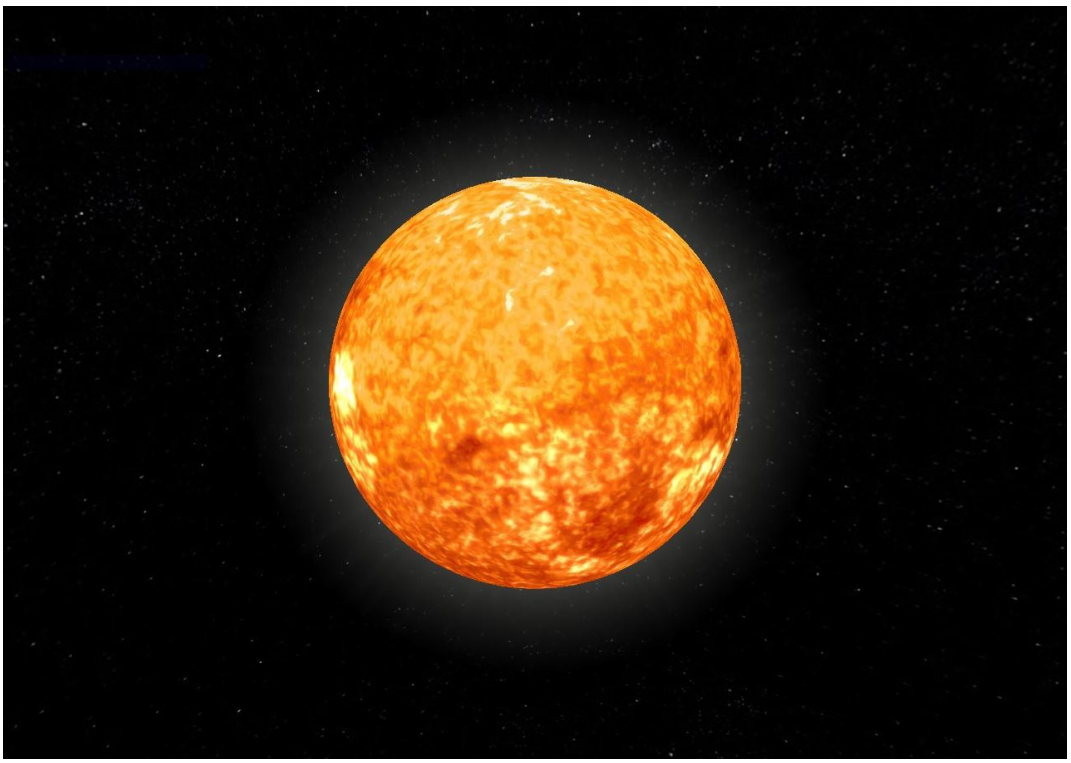

Рис. 8. Добавление свечения от Солнца.

Также стоит отметить, что в функции `setSun()` внутрь Солнца добавляется источник света `THREE.PointLight`. Он нужен для того, чтобы освещать планеты и спутники.

4.4. Анимация планеты

После добавления Солнца необходимо добавить планеты. Добавление планеты на сцену происходит также, как и в случае с добавлением Солнца, то есть сначала загружается текстура и затем вызывается функция `getSphere()`. Также для наглядности орбита планеты отображается в виде кольца с тонкой линией. Создание орбиты происходит в функции `getOrbit()`. Пример планеты Земля и её орбиты показан на рисунке 9.

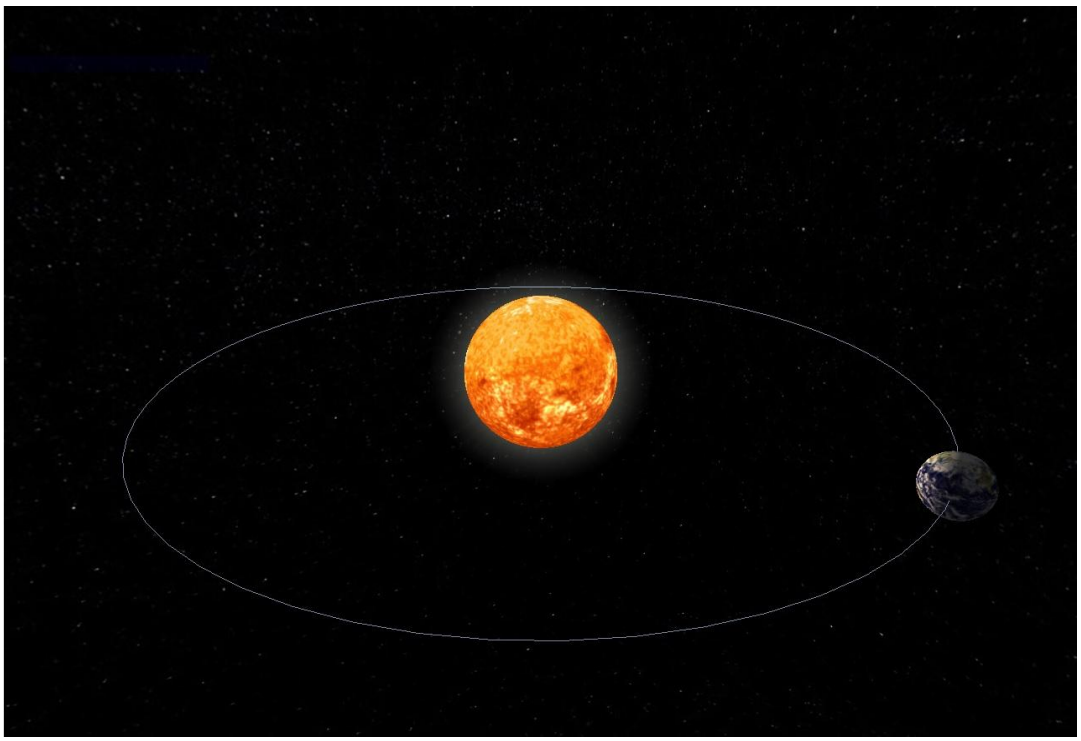

Рис. 9. Солнце, Земля и её орбита.

Для того, чтобы анимировать планету необходимо часто рендерить сцену, то есть обновлять кадр, который видит пользователь. С этой целью используется функция `update()`. В ней происходит отрисовка сцены (`renderer.render(scene, camera)`) каждый раз, когда обновляется экран (на типичном экране это происходит 60 раз в секунду). Для перемещения планеты в пространстве необходимо менять её координаты. Функция `movePlanet()` делает именно это. Код `movePlanet()`:

```
function movePlanet(myPlanet, myData, myTime, displacementX=0,
displacementZ=0, stopRotation) {
      if (orbitData.runRotation && !stopRotation) {
            myPlanet.rotation.y += myData.rotationRate;
      }
      if (orbitData.runOrbit) {
            myPlanet.position.x = Math.cos(myTime
            * (1.0 / (myData.orbitRate * orbitData.value)))
            * myData.distanceFromAxis;
            myPlanet.position.z = Math.sin(myTime
            * (1.0 / (myData.orbitRate * orbitData.value)))
            * myData.distanceFromAxis;
      }
      myPlanet.position.x += displacementX;
      myPlanet.position.z += displacementZ;
}
```

Изменение `myPlanet.rotation.y` позволяет вращать планету вокруг своей оси, а `myPlanet.position.x` и `myPlanet.position.z` меняют координаты x и z соответственно.

4.5. Создание графического интерфейса пользователя

Для управления некоторыми свойствами при помощи библиотеки `dat.gui.js` был добавлен простой графический интерфейс. Его инициализация происходит в функции `setGUI()`, код которой представлен ниже:

```
function setGUI(){
	var gui = new dat.GUI();
	gui.width = 350;
	var folder1 = gui.addFolder('Свет');
	folder1.add(pointLight, 'intensity', 0,
	2).name('Интенсивность');
	folder1.add(ambientLight, 'intensity', 0, 1).name('Освещение
	объектов');
	var folder2 = gui.addFolder('Движение планет');
	folder2.add(orbitData, 'value', 0, 200).name('Период
	обращения');
	folder2.add(orbitData, 'runOrbit', 0, 1).name('Остановить
	движение');
	folder2.add(orbitData, 'runRotation', 0, 1).name('Остановить
	вращение');
	var folder3 = gui.addFolder('Видимость орбит');
	folder3.add(visibleOrbit, 'visible').name('Показать');
	folder3.add(visibleOrbit, 'hidden').name('Скрыть');
}
```

На рисунке 10 можно увидеть графический интерфейс.

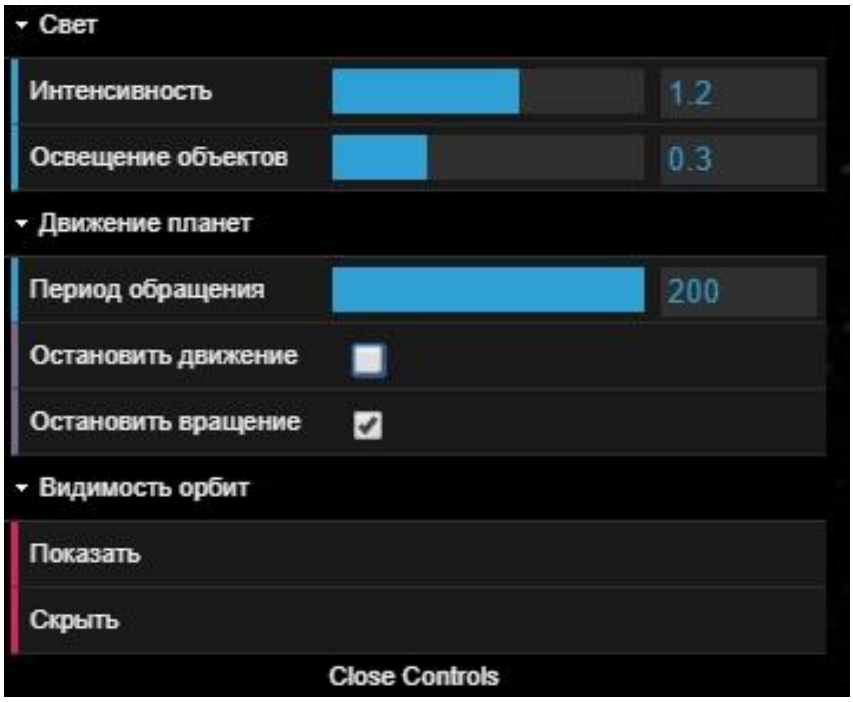

Рис. 10. Графический интерфейс пользователя.

Коротко о каждой переменной:

- Свет → Интенсивность - яркость освещения планет Солнцем
- Свет → Освещение объектов - яркость глобального освещения планет
- Движение планет → Период обращения - скорость движения планет по орбите
- Движение планет → Остановить движение - флажок для остановки/продолжения движения планет по орбите
- Движение планет → Остановить вращение - флажок для остановки/продолжения вращения планет
- Видимость орбит → Показать - показать линии орбит
- Видимость орбит → Скрыть - скрыть линии орбит

4.6. Возможность рассмотрения планет вблизи

Изначально камера сфокусирована на Солнце и для того, чтобы рассмотреть какой-либо другой объект солнечной системы необходимо менять фокус камеры на этот объект. Чтобы поменять фокус камеры на объект нужно знать координаты этого объекта в пространстве. Для этой цели

в Three.js используется Raycaster. Он получает координаты объекта, над которым в данный момент находится курсор мыши. Ниже приведён код инициализирующий Raycaster:

```
function setRaycaster(){
      raycaster = new THREE.Raycaster();
      renderer.domElement.addEventListener( 'click', raycast,
      false );
      renderer.domElement.addEventListener( 'mousemove', raycast2,
      false );
}
```

Здесь определены два обработчика событий:

- клик по объекту - меняет фокус камеры на объект
- наведение курсора мыши на объект - отображение информации об объекте

На рисунке 11 изображен пример обоих событий.

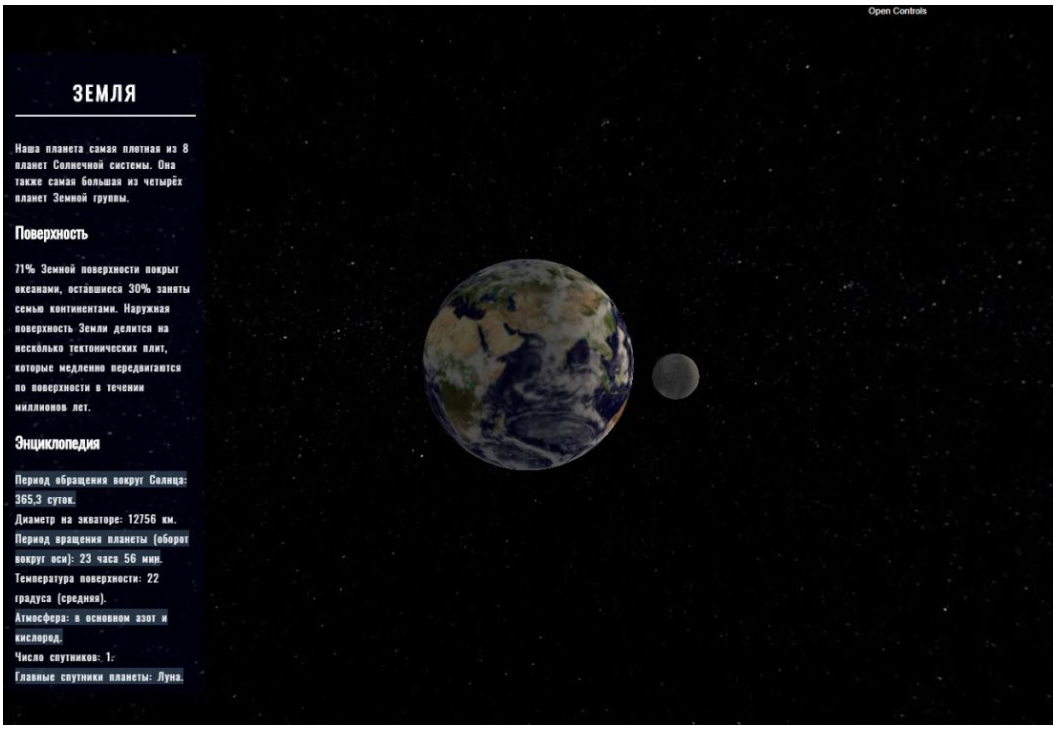

Рис. 11. Просмотр планеты вблизи.

5. Тестирование приложения

В результате курсовой работы было разработано веб-приложение для демонстрации модели солнечной системы. Исходный код можно найти в репозитории https://github.com/1uckyy/SolarSystem. Также само веб-приложение можно опробовать по ссылке https://1uckyy.github.io/.

На рисунках 12 и 13 показаны скриншоты финального приложения.

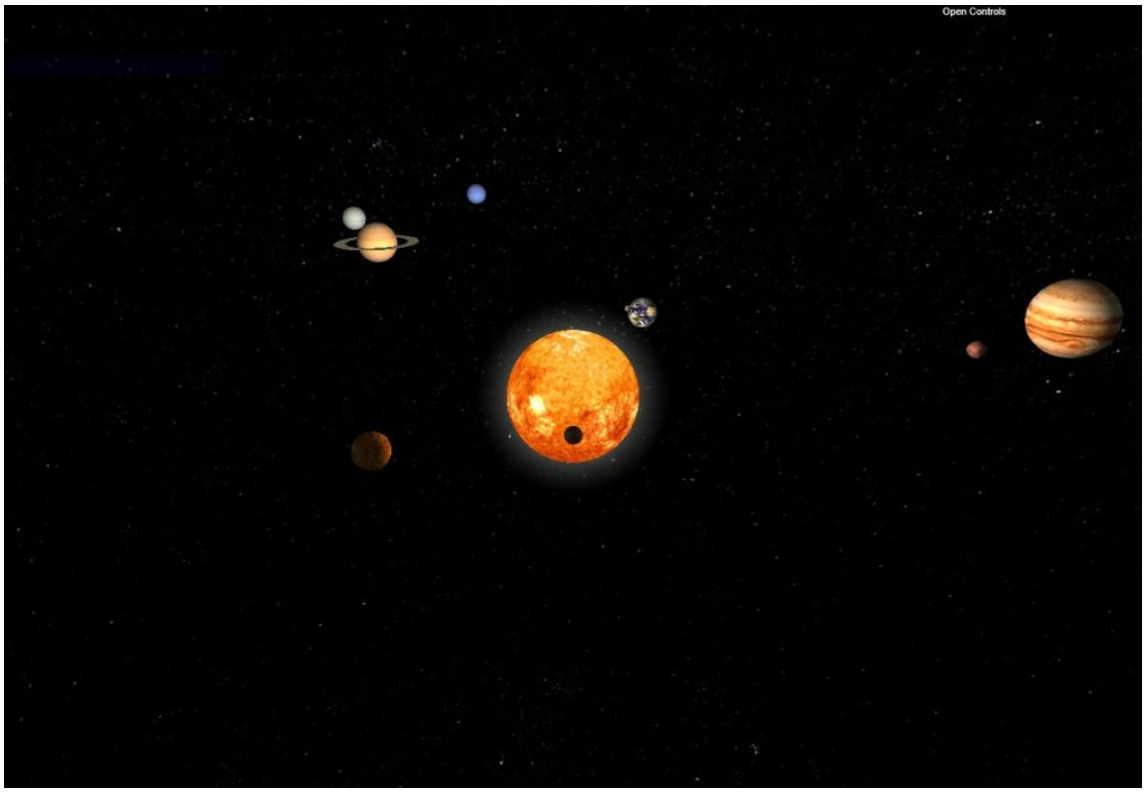

Рис. 12. Сцена финального приложения.

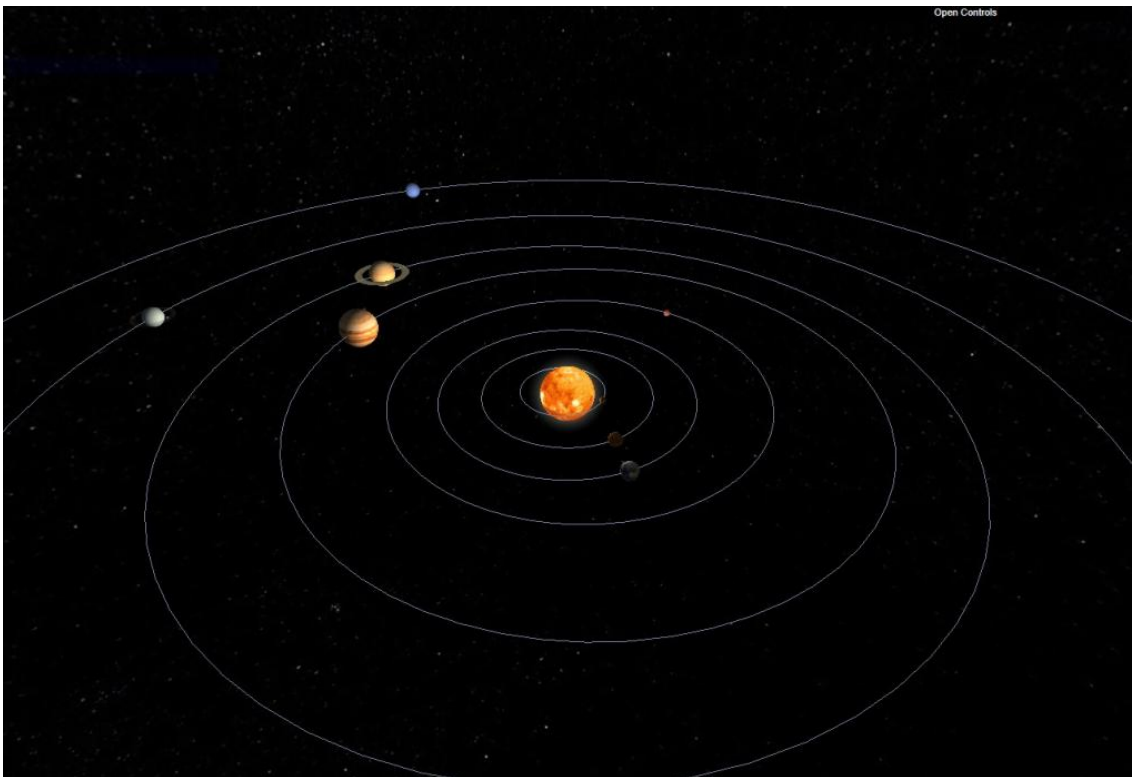

Рис. 13. Сцена финального приложения.

Так как разработанное веб-приложение для рендеринга сцены использует ресурсы видеокарты, то для проверки производительности необходимо протестировать именно графический процессор. Тестирование осуществлялось на 3-х разных видеокартах: Nvidia GTX 1050 (компьютер), Intel HD Graphics (компьютер), Nvidia 940MX (ноутбук). В результате теста удалось определить среднее количество кадров в секунду для каждой из карт:

- Nvidia GTX 1050: 60 кадров в секунду
- Intel HD Graphics: 60-57 кадров в секунду
- Nvidia 940MX: 60-55 кадров в секунду

Результаты теста показали, что приложение стабильно обрабатывает комфортные для человека 60 кадров в секунду на любой современной видеокарте.

Производительность памяти:

Для теста памяти веб-приложение было открытым в течении нескольких часов и в результате снятия показаний с диспетчера задач, который встроен в Google Chrome (Рис. 14), оказалось, что нагрузка на

память не изменилась. Это означает, что веб-приложение не имеет утечек памяти и работает очень хорошо. На нижеследующем рисунке показан объем памяти рабочего стола и памяти графического процессора, который использует сцена.

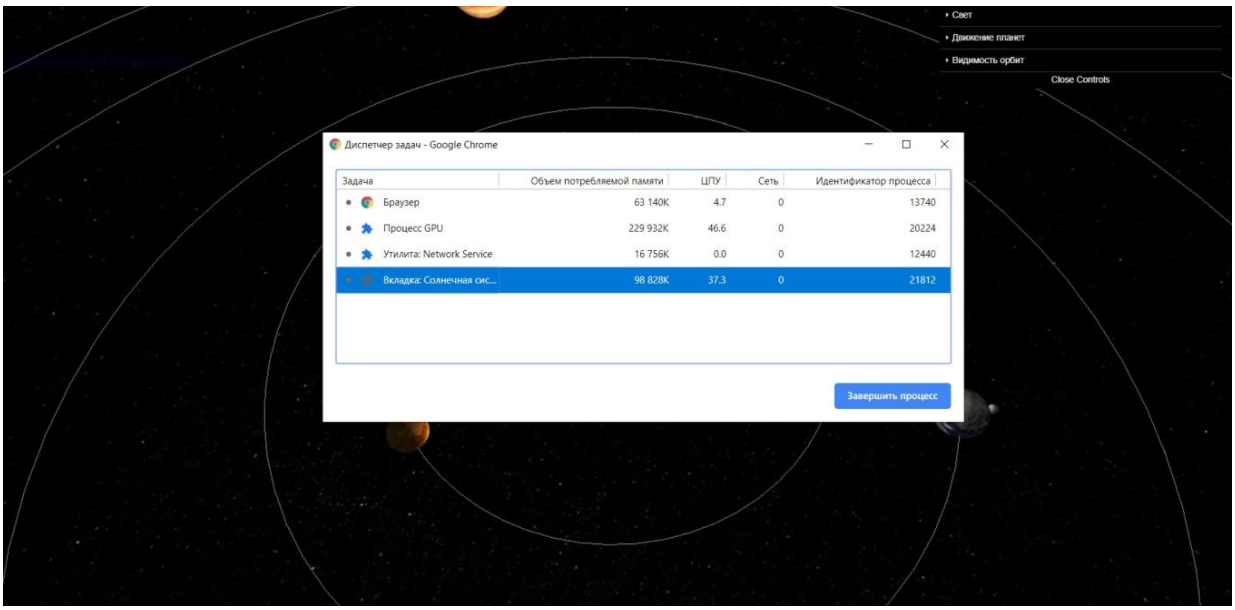

Рис. 14. Диспетчер задач Google Chrome.

Заключение

В результате выполнения курсового проекта были выполнены следующие задачи:

- разработано веб-приложение для демонстрации модели солнечной системы, в котором имеются планеты, орбиты планет, Солнце и Луна и каждая планета и Луна движутся по своей орбите и вращаются вокруг своей оси;
- разработан графический интерфейс, который позволяет менять некоторые параметры (скорость обращения планет, интенсивность света от Солнца и другие);
- реализована возможность рассмотрения планет и Луны более подробно, при этом получая текстовую информацию об объекте.

Веб-приложение бесплатно и может использоваться кем-либо для получения более подробного представления о солнечной системе.

Также было проведено тестирование веб-приложения, в ходе которого выяснилось, что большинство современных компьютеров легко справляются с нагрузкой, связанной с работой приложения.

Список литературы

1. Мацуда, К. Webgl: программирование трехмерной графики: пер. с англ. / К. Мацуда, Р. Ли. - М.: ДМК Пресс, 2015. - 494 с.
2. Дирксен Й. Learning Three.js: The JavaScript 3D Library for WebGL: пер. с англ. / Й. Дирксен. - М.: ДМК Пресс, 2013. - 453 с.
3. Вильданов А. Н. 3D-моделирование на WebGL с помощью библиотеки Three.js / А. Н. Вильданов. - Уфа: РИЦ БашГУ, 2014. - 114с.