**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Владимирский государственный университет**

**имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(ВлГУ)**

Кафедра информационных систем и программной инженерии

**Курсовой проект**

по дисциплине «Программирование графических приложений»

**РАЗРАБОТКА ГРАФИЧЕСКОГО ВЕБ-ПРИЛОЖЕНИЯ**

**С ИСПОЛЬЗОВАНИЕМ WEBGL**

Предметная область:

«Склад»

Выполнил: студент группы ПРИм-121

Евстигнеев Андрей Васильевич

Проверил: профессор кафедры ИСПИ

Жигалов Илья Евгеньевич

Владимир 2023

АННОТАЦИЯ

Курсовой проект содержит 39 страниц, 6 рисунков, 11 источников литературы.

Курсовой проект выполнен по дисциплине «Программирование графических приложений». Темой курсового проекта является разработка графического веб-приложения с использованием WebGL в предметной области «Склад». Сцена содержит модели основных объектов и ландшафт, близкие по стилистике к тематике, а также включает вспомогательные объекты.

ABSTRACT

The course project is contained 39 pages, 6 illustrations, 11 bibliography.

The course project was carried out in the discipline "Programming of graphical applications". The topic of the course project is the development of a graphical web application using WebGL in the subject area "Warehouse". The scene contains models of the main objects and the landscape, similar in style to the theme, and also includes auxiliary objects.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc130933001)

[1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ 6](#_Toc130933002)

[1.1 Введение 6](#_Toc130933003)

[1.2 Описание 8](#_Toc130933004)

[1.3 Цель разработки 8](#_Toc130933005)

[1.4 Функциональные требования 9](#_Toc130933006)

[1.5 Нефункциональные требования 9](#_Toc130933007)

[1.6 Инструменты разработки 9](#_Toc130933008)

[1.7 Интерфейс приложения 10](#_Toc130933009)

[1.8 Требования к программной документации 10](#_Toc130933010)

[1.9 Стадии и этапы разработки 11](#_Toc130933011)

[2 ЦЕЛИ И ЗАДАЧИ ПРОЕКТА 12](#_Toc130933012)

[3 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 13](#_Toc130933013)

[3.1 Моделируемая сцена 13](#_Toc130933014)

[3.2 Моделируемые объекты 13](#_Toc130933015)

[3.3 Моделируемые события 14](#_Toc130933016)

[4 ПРОЕКТИРОВАНИЕ 15](#_Toc130933017)

[4.1 Описание архитектуры 15](#_Toc130933018)

[4.2 Описание технологий 15](#_Toc130933019)

[4.3 Описание реализации 16](#_Toc130933020)

[5 РАЗРАБОТКА АЛГОРИТМА И ДИЗАЙНА ПРИЛОЖЕНИЯ 18](#_Toc130933021)

[5.1 Разработка схемы алгоритма программы 18](#_Toc130933022)

[5.2 Разработка дизайна приложения 18](#_Toc130933023)

[6 МОДЕЛИРОВАНИЕ ГРАФИЧЕСКИХ ОБЪЕКТОВ 19](#_Toc130933024)

[6.1 Выбор и обоснование моделей 19](#_Toc130933025)

[6.2 Создание объектов с помощью WebGL 19](#_Toc130933026)

[6.3 Создание объектов с помощью графических редакторов 20](#_Toc130933027)

[7 МОДЕЛИРОВАНИЕ СЦЕНЫ 21](#_Toc130933028)

[7.1 Использование материалов, текстур, освещения, цветовых моделей 21](#_Toc130933029)

[7.2 Создание анимации и визуальных эффектов 21](#_Toc130933030)

[8 МОДЕЛИРОВАНИЕ СОБЫТИЙ 22](#_Toc130933031)

[8.1 Логика работы приложения 22](#_Toc130933032)

[8.2 Реализация взаимодействия объектов 22](#_Toc130933033)

[9 ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ И УПРАВЛЕНИЕ ПРИЛОЖЕНИЕМ 23](#_Toc130933034)

[9.1 Обоснование и реализация дизайна интерфейса пользователя 23](#_Toc130933035)

[9.2 Проектирование интерактивного управления сценой 23](#_Toc130933036)

[9.3 Настройки приложения 23](#_Toc130933037)

[10 ПРОГРАММИРОВАНИЕ ПРИЛОЖЕНИЯ 25](#_Toc130933038)

[10.1 Отладка и оптимизация кода 25](#_Toc130933039)

[10.2 Тестирование приложения, анализ программного продукта 25](#_Toc130933040)

[11 РАЗРАБОТКА ПРОГРАММНОЙ ДОКУМЕНТАЦИИ 26](#_Toc130933041)

[ЗАКЛЮЧЕНИЕ 27](#_Toc130933042)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 28](#_Toc130933043)

[ПРИЛОЖЕНИЯ 29](#_Toc130933044)

[Тексты программ с комментариями: 29](#_Toc130933045)

[Полученные результаты (скриншоты) 35](#_Toc130933046)

[Программная документация 38](#_Toc130933047)

ВВЕДЕНИЕ

Для отображения 3D-объектов в браузере есть большое количество технологий, самые популярные - Unity и Unreal. У них есть как свои плюсы, так и очевидные минусы. Необходимость установки приложения, очень медленная загрузка и работа, повышенные требования к “железу” и т.д.

Технология WebGL поддерживается всеми современными браузерами и лишена этих недостатков, она отлично подходит для отображения 3D-сцен на сайтах. Перечислим основные плюсы данного фреймворка:

* WebGL-проекты одинаково хорошо работают во всех браузерах и на всех устройствах;
* Не требует значительных вычислительных мощностей, старенький ноутбук вполне подойдет;
* Для работы приложения не требуется установка дополнительных плагинов или драйверов;
* Приложения, разработанные на WebGL, не требуют установки и работают онлайн;
* Легкая интеграция с любым сайтом, интернет-магазином или CRM-системой;
* Высокое качество графики - реалистичные текстуры, освещение и динамические эффекты;

Данная библиотека используется в различных сферах, за счет её доступности пишутся огромное количество приложений на её основе, например, в автопромышленности, игровой, а также в архитектуре.

1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1.1 Введение

В настоящее время возможности web-браузеров позволяют отображать интерактивную трехмерную графику без подключения плагинов и сторонних приложений (например, технология AdobeFlash). Технология WebGL сделала возможным создание web-приложений, работающих с интерактивной трехмерной графикой, применяя язык программирования JavaScript. WebGL представляет собой библиотеку для языка программирования JavaScript.

WebGL представляет собой технологию, базирующуюся на OpenGL ES 2.0 и предназначенную для рисования и отображения интерактивной 2D- и 3D-графики в веб-браузерах. При этом для работы с данной технологией не требуются сторонние плагины или библиотеки. Вся работа веб-приложений с использованием WebGL основана на коде JavaScript, а некоторые элементы кода – шейдеры могут выполняться непосредственно на графических процессорах, на видеокартах, благодаря чему разработчики могут получить доступ к дополнительным ресурсам компьютера, увеличить быстродействие. Таким образом, для создания приложений разработчики могут использовать стандартные для веб-среды технологии HTML/CSS/JavaScript и при этом также применять аппаратное ускорение графики.

Актуальность выбранной темы, 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;
* поскольку WebGL для рендеринга графики использует графический процессор на видеокарте (GPU), то для этой технологии характерна высокая производительность, которая сравнима с производительностью нативных приложений.

**Глоссарий**

|  |  |
| --- | --- |
| Определения, обозначения, сокращения | Расшифровка |
| Web-разработка | Процесс создания веб-сайта или веб-приложения |
| WebGL | Контекст элемента canvas HTML, который обеспечивает API 3D графики без использования плагинов |
| Сцена | Своеобразная платформа, где размещаются все объекты, которые мы создаем |
| Графический объект | Двухмерные изображения с элементами объема, который придается за счет работы с освещением и другими элементами, создающими на экране визуальную иллюзию |
| ПГП | Дисциплина "Программирование графических приложений" |
| Анимация | Процесс придания способности двигаться и/или видимости жизни объектам и мёртвым телам |

1.2 Описание

Основная цель выполнения курсового проекта (КП) по ПГП состоит в освоении средств разработки веб-приложений с использованием технологии WebGL.

Предметом курсового проекта является разработка программы, формирующей заданную трехмерную графическую сцену с интерактивным управлением ею. При этом используются возможности JavaScript и WebGL по созданию графических приложений, формированию простых графических объектов из примитивов, применению геометрических преобразований, использованию цветов, текстур и освещения со специальными эффектами, пиксельных операций, построению теней, анимации сцены и управлению сценой.

1.3 Цель разработки

Целью курсовой работы является разработка web-приложения на языке программирования JavaScript, функциональным назначением которого является выбор дизайна интерьера склада.

Поставленная цель определила постановку следующих задач:

– формирование пользовательского интерфейса;

– изучение технологии WebGL;

– изучение и совершенствование знаний HTML 5, JavaScript;

– разработка алгоритма создания сцены;

– моделирование объектов;

– изучение предметной области;

1.4 Функциональные требования

Сцена должна содержать модели основных объектов и фон (ландшафт), близкие по стилистике к тематике, а также включать любые вспомогательные объекты. Обязательными являются использование не менее двух управляемых пользователем объектов, взаимодействие объектов, неуправляемая циклическая анимация.

Сцена должна иметь возможность настройки (освещение, визуальные эффекты) и управления (изменение положения наблюдателя, управление положением и визуальными свойствами объектов, переключение управления между объектами), использовать взаимодействие объектов.

1.5 Нефункциональные требования

Система должна быть реализована для систем Windows. Должна быть возможность использовать английский язык. Система должна быть доступной и гибкой к изменениям. До релизной версии продукта доступ к системам должна иметь только группа тестирования и разработчики.

1.6 Инструменты разработки

Для разработки данной системы в первую очередь необходим для использования браузер. Самых два комфортных для веб-разработки браузера являются Google Chrome и Mozilla Firefox. Нашим выбором является Google Chrome, так как он является наиболее популярным.

Для того, чтобы создавать сложный обновляемый контент на странице, мы выберем для реализации язык программирования JavaScript. Он реализует на стороне клиента большую часть динамики и позволит каждому пользователю управлять системой самостоятельно.

Так как в нашей системе стоит задача реализации 3D-сцен, нам стоит обратиться к библиотеке для JavaScript – WebGL. Она реализует 2D- и 3D-рендеринг. Сама библиотека основана на OpenGL ES, который является довольно распространенным API для различных языков программирования.

1.7 Интерфейс приложения

Система представляет собой окно браузера, в котором отрабатывает скрипт. Во вкладке браузера должен быть отображен ландшафт. На экране должны быть расположены кнопки «Вращение влево», «Вращение вправо».

На ландшафте должна быть расположена модель со вспомогательными объектами. Должны быть кнопки, которые отвечают за «Zoom». А также кнопка «Анимация», по нажатию на которую начиналась бы анимация, а сама кнопка менялась бы на кнопку «остановить анимацию».

1.8 Требования к программной документации

В состав программной документации предварительно можно включить:

1. Техническое задание
2. Код программы
3. Пояснительную записку
4. Описание применения
5. Руководство пользователя

1.9 Стадии и этапы разработки

1. Разработка технического задания

2. Изучение предметной области

3. Проектирование и разработка визуальной составляющей системы

4. Моделирование сцены, событий, графических объектов

5. Разработка приложения и интерфейса

6. Тестирование приложения

7. Написание документации

2 ЦЕЛИ И ЗАДАЧИ ПРОЕКТА

Цель проекта - разработать интерактивную трехмерную графическую сцену склада с заданными параметрами.

Задачи проекта:

1. Создание 3D-модели склада с различными элементами.

2. Реализация интерактивного управления этой моделью.

3. Использование возможностей JavaScript и WebGL для создания графических приложений.

4. Формирование простых графических объектов из примитивов.

5. Работа с цветом, текстурами, освещением и тенями.

6. Анимация сцены.

7. Реализация управления сценой.

8. Использование библиотек, таких как Three.js, для упрощения процесса разработки.

9. Документирование проекта.

10. Создание понятного пользовательского интерфейса для управления сценой.

# 3 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

3.1 Моделируемая сцена

Моделируемая сцена — это склад, на котором расположены различные объекты. В центре склада находится большая бочка с дымом, которая может быть выполнена в виде трехмерной модели. Вокруг бочки расположены ящики разных размеров и форм, а также шиферные листы. На складе также могут быть другие объекты, такие как инструменты, мешки с грузом и т.д. Вся сцена должна быть выполнена в трехмерном формате и иметь возможность интерактивного управления.

3.2 Моделируемые объекты

В проекте моделируются объекты, связанные со складом, такие как:

1. Большая бочка с дымом.

2. Ящики.

3. Ведро с водой.

4. Материалы для стройки, например, шифер.

5. Плоскости пола и стен.

Все эти объекты будут созданы в трехмерном формате и будут иметь возможность интерактивного управления.

3.3 Моделируемые события

В проекте предусмотрены различные события, которые могут происходить в моделируемой сцене. Например, пользователь может выбирать и перемещать объекты на складе кнопками клавиатуры или мышкой. В проекте также предусмотрена возможность анимации объектов, например, перемещение бочки с дымом с одного места на другое. Кроме того, пользователь может выбирать различные виды камеры для просмотра сцены, а также изменять угол обзора. Все эти события будут обрабатываться в реальном времени и отображаться на экране в трехмерном формате.

# 4 ПРОЕКТИРОВАНИЕ

4.1 Описание архитектуры

Клиентская часть приложения будет написана на JavaScript, используя WebGL и Three.js для создания 3D-графики в браузере. Для упрощения создания 3D-графики в браузере используется библиотека Three.js. Клиентская часть приложения будет работать в браузере пользователя и будет обрабатывать все запросы и действия пользователя. В данном случае серверная часть приложения не требуется, так как все данные и логика будут обрабатываться на клиентской стороне.

Архитектура проекта будет разделена на несколько слоев. На самом нижнем уровне будет находиться слой работы с графикой, который будет отвечать за создание и отображение трехмерных объектов на экране. На следующем уровне будет находиться слой управления, который будет обрабатывать пользовательские действия и изменять состояние сцены. На верхнем уровне будет находиться слой представления, который будет отвечать за отображение пользовательского интерфейса и взаимодействие с пользователем.

Таким образом, архитектура проекта будет модульной и масштабируемой, что позволит нам легко добавлять новые функции и улучшать производительность.

4.2 Описание технологий

JavaScript — это язык программирования, который используется для создания интерактивных веб-страниц и приложений. Он может быть использован для создания динамических элементов на странице, обработки событий и многого другого.

WebGL — это технология, которая позволяет создавать 3D-графику в браузере. Она основана на языке программирования OpenGL и позволяет создавать высококачественные 3D-изображения и анимацию.

Three.js — это библиотека JavaScript, которая упрощает создание 3D-графики в браузере. Она предоставляет множество инструментов для создания и управления 3D-объектами, а также для создания эффектов и анимации.

4.3 Описание реализации

1. Создание HTML-страницы, на которой будет размещено приложение. На этой странице мы подключаем необходимые библиотеки и скрипты, а также создаем контейнер для 3D-сцены.

2. Использование JavaScript для создания и настройки 3D-сцены. Используем Three.js для создания и управления 3D-объектами, добавления текстур и материалов, настройки освещения и т.д.

3. Использование WebGL для отображения 3D-сцены в браузере. WebGL позволяет использовать аппаратное ускорение для отображения 3D-графики, что обеспечивает высокую производительность и качество изображения.

4. Добавление интерактивности в приложение, используя JavaScript. Добавляем обработчики событий для пользовательского ввода, такие как нажатие кнопок мыши или клавиатуры, и используем их для управления 3D-объектами или другими элементами приложения.

5. Оптимизация производительности приложения, используя различные техники, таких как сокращение числа полигонов, использование LOD (уровней детализации), оптимизация текстур и т.д.

5 РАЗРАБОТКА АЛГОРИТМА И ДИЗАЙНА ПРИЛОЖЕНИЯ

5.1 Разработка схемы алгоритма программы

1. Создание сцены Three.js, на которой будут размещены объекты.

2. Создание и размещение геометрических фигур для объектов (ящиков, бочки, ведра воды, шифера и дыма) с помощью Three.js.

3. Создание материалов для каждого объекта.

4. Размещение объектов на сцене в соответствии с их позицией на складе.

5. Создание источника света для имитации освещения склада.

6. Добавление возможности перемещения камеры для просмотра склада из разных углов.

7. Обработка взаимодействия пользователя с объектами (например, при нажатии на ящик, он может изменять свое положение).

8. Обработка анимации для объектов (например, анимация дыма).

5.2 Разработка дизайна приложения

При разработке дизайна приложения соблюдалась общая цветовая гамма приглушённых цветов, соответствующих более реалистичному содержанию сцены. Пользовательский интерфейс довольно упрощён и отражает необходимость наличия минимально необходимого отображения для всех интерактивных элементов.

# 6 МОДЕЛИРОВАНИЕ ГРАФИЧЕСКИХ ОБЪЕКТОВ

6.1 Выбор и обоснование моделей

1. Большая бочка с дымом - такой объект может быть использован в складе, связанном с хранением и транспортировкой жидкостей, например, химических веществ или пищевых ингредиентов.

2. Ящики - являются одним из наиболее распространенных объектов на складах.

3. Материалы для стройки (например, шифер) - такие объекты могут быть использованы в складе, связанном с хранением и транспортировкой различных материалов и инструментов.

4. Ведро с водой - такой объект может быть использован на складе для уборки или на случай протечки крыши.

6.2 Создание объектов с помощью WebGL

С помощью WebGL мы используем кубические формы для создания ящиков, поверхность вращения с комбинированной образующей для бочки, а для ведра воды - цилиндрическую форму.

Чтобы объекты выглядели более реалистично, мы добавляем к ним текстуры и материалы. Мы также используем свет и тени, чтобы создать более реалистичную атмосферу.

Для управления движением объектов на складе мы используем функции Three.js. Мы изменяем позицию объектов, а также добавляем анимации и эффекты, чтобы сделать программу более интерактивной и интересной для пользователей. В целом, создание объектов в Three.js позволяет создавать привлекательные и интерактивные 3D-модели для нашего склада.

6.3 Создание объектов с помощью графических редакторов

Одним из элементов сцены является ящик, модель которого импортирована из Blender со всеми настройками. В целом использование Blender в связке с Three.js многократно упростило создание сложных составных фигур и предоставило возможность потратить сэкономленное время на реализацию анимации и взаимодействия с объектами сцены.

# 7 МОДЕЛИРОВАНИЕ СЦЕНЫ

7.1 Использование материалов, текстур, освещения, цветовых моделей

Для реализации объектов были использованы текстуры ящика, кирпичных стен, бетонного пола. Из материалов можно выделить имитацию полупрозрачного стекла для окон и металла для ведра.

7.2 Создание анимации и визуальных эффектов

В программе реализована анимация дыма, состоящего из множества частиц, двигающихся в заданном диапазоне. Дым относится к объекту бочки и двигается относительно её верхнего края.

Помимо дыма на сцене имеется анимация движения капель, падающих по определённой полосе с заданной частотой. Также реализован эффект разбрызгивания капли во все стороны при падении на пол.

# 8 МОДЕЛИРОВАНИЕ СОБЫТИЙ

8.1 Логика работы приложения

Мы импортировали ящик из Blender в Three.js и создали остальные объекты на складе, используя геометрические формы. Создали сцену, где разместили все объекты. Мы установили параметры камеры, света, теней и фонового изображения, чтобы получить нужную атмосферу и внешний вид. Добавили каждый объект на сцену с помощью метода add. Чтобы объекты выглядели реалистично, мы изменили их позицию, поворот и масштаб на сцене. Мы создали функции для перемещения объектов с помощью мыши или клавиатуры, чтобы пользователи могли взаимодействовать с приложением. Добавили анимации и эффекты, чтобы сделать программу более интерактивной и захватывающей для пользователей. Например, на сцене есть игра крестики-нолики, в которую пользователь играет против компьютера. Также есть возможность ловить капли с помощью ведра и увеличивать свой счёт. Мы настроили свойства отображения объектов, такие как текстуры, материалы и освещение, чтобы сделать их более реалистичными. Отобразили объекты на сцене, чтобы пользователи могли их увидеть. Приложение готово для использования и предоставляет пользователю увлекательную визуальную обстановку для изучения склада.

8.2 Реализация взаимодействия объектов

Взаимодействие объектов реализовано в виде капель, ударяющихся об пол и разбрызгивающихся в противоположном направлении.

# 9 ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ И УПРАВЛЕНИЕ ПРИЛОЖЕНИЕМ

9.1 Обоснование и реализация дизайна интерфейса пользователя

Реализация визуального интерфейса осуществлена в минималистичном стиле с помощью блоков текста поверх Canvas. Их размер достаточно большой для того, чтобы пользователь не упустил их из виду и смог разобраться, где отображается результат взаимодействия с объектами в двух интерактивных частях склада.

9.2 Проектирование интерактивного управления сценой

Большая часть взаимодействия со сценой осуществляется с помощью кнопок:

* 1, 2, 3, 4, 5, 6, 7, 8, 9 для игры в «крестики-нолики»
* A, W, S, D для передвижения бочки
* Z, X для передвижения ведра

9.3 Настройки приложения

При написании кода тщательно подбирались коэффициенты для настройки различных параметров вроде скорости падения капель, наиболее эффектного угла падения света на сцену.

Осуществили тестирование приложения на различных устройствах и браузерах, чтобы убедиться, что оно работает корректно и оптимально на всех платформах. Установили наиболее подходящую версию Three.js и оптимизировали функции для менее затратного по ресурсам отображения элементов сцены.

# 10 ПРОГРАММИРОВАНИЕ ПРИЛОЖЕНИЯ

10.1 Отладка и оптимизация кода

Мы отладили и оптимизировали код приложения, чтобы оно работало более эффективно, проанализировали производительность приложения и определили проблемные места, после чего провели рефакторинг кода для устранения этих проблем. В результате приложение работает более быстро и эффективно, что улучшает пользовательский опыт и повышает качество итогового приложения.

10.2 Тестирование приложения, анализ программного продукта

Мы использовал различные методы тестирования, включая модульное тестирование, функциональное тестирование, тестирование производительности и пользовательское тестирование. Проверили, как приложение работает в различных условиях и на различных устройствах, чтобы убедиться, что оно не имеет ошибок и сбоев. В результате обнаружили неправильно отображающийся блок текста на маленьких экранах. Также обнаружили проблему падения фреймрейта при срабатывании дублирующейся функции, после чего перенастроили метод её использования.

# 11 РАЗРАБОТКА ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

Программная документация для проекта состоит из следующих элементов:

1. Описание проекта: описание приложения, его основных функций и возможностей.

2. Техническое описание: описание используемых технологий и инструментов, включая Three.js, Blender, HTML, CSS и JavaScript.

3. Архитектура приложения: описание структуры приложения, его компонентов и способов их взаимодействия.

4. Описание функций: описание функций, используемых в приложении.

5. Интерфейс пользователя: описание пользовательского интерфейса, его элементов управления и способов взаимодействия с приложением.

6. Тестирование: описание тестов, проведенных для проверки работоспособности приложения и выявления ошибок.

7. Описание процесса разработки: описание этапов разработки приложения, его компонентов и функций, а также принятых решений и проблем, с которыми столкнулся разработчик.

# ЗАКЛЮЧЕНИЕ

В результате выполнения проекта мы получили опыт в разработке веб-приложений с использованием технологий Three.js, Blender, HTML, CSS и JavaScript. Вызвало научный и практический интерес работать над созданием 3D-графики, разработкой пользовательского интерфейса и взаимодействием с пользователем, используя актуальные технологии и инструменты. Также были улучшены навыки тестирования и отладки программного обеспечения.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Вильданов А.Н. 3D-моделирование на WebGL с помощью библиотеки Three.js. - Уфа: РИЦ БашГУ, 2014. - 113 с. ISBN 987-5-7477-3560-6
2. Жигалов И.Е. Компьютерная графика: Курс лекций/ Владим. гос. ун-т. Владимир, 2004. 124 с. ISBN 5-89368-459-1.
3. Жигалов И.Е., Новиков И.А. Программирование компьютерной графики. Учебное пособие./ Владим. гос. ун-т. Владимир, 2014. 96 с. ISBN 978-5-9984- 0437-5.
4. Жигалов И.Е., Новиков И.А. Программирование двухмерной компьютерной графики. Учебное пособие./ Владим. гос. ун-т. Владимир, 2015. 120 с. ISBN 978-5-9984-0610-2.
5. Жигалов И. Е., Новиков И.А. Программирование трехмерной компьютерной графики : учеб. пособие / Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. ‒Владимир: Изд-воВлГУ, 2016. ‒92 с. ISBN 978-5-9984-0685-0.
6. Мацуда К., Ли Р. WebGL: программирование трехмерной графики. / Пер. с англ. - М.: ДМК Пресс, 2015. - 494 с.: ил. ISBN 978-5-97060-146-4
7. Порев В.Н. Компьютерная графика.–СПб.: БХВ-Петербург, 2002.- 432 с.
8. Основы Javascript. [Электронный ресурс] - Режим доступа: <https://learn.javascript.ru/first-steps>.
9. Херн Д. Компьютерная графика и стандарт OpenGL - Москва: Вильямс, 2005 .- 1158 c., - ISBN 5-8459-0772-1.
10. Stemkoski, Lee. Three.js - examples [Электронный ресурс] - Режим доступа: <http://stemkoski.github.io/Three.js>.
11. Three.js - JavaScript 3D library [Электронный ресурс] - Режим доступа: <https://threejs.org/docs/index.html>.

ПРИЛОЖЕНИЯ

Тексты программ с комментариями:

//код для генерации помещения

let generateTheWall = (rotationX, rotationY, positionX, positionY, positionZ, textureSrc, windows) => {

        var windowGeometry = new THREE.BoxGeometry(60, 120, 2);

        var windowMaterial = new THREE.MeshPhongMaterial({ color: 0x628090 });

        var septumTexture = new THREE.ImageUtils.loadTexture('assets/wood.jpg');

        septumTexture.minFilter = THREE.NearestFilter;

        var septumGeometry = new THREE.BoxGeometry(10, 140, 2);

        var septumMaterial = new THREE.MeshPhongMaterial({ map: septumTexture });

        let generateTheSeptum = (x, y, horizontal) => {

            var septumMesh = new THREE.Mesh(septumGeometry, septumMaterial);

            if (horizontal) {

                septumMesh.rotation.z = Math.PI / 2;

            }

            septumMesh.position.set(x, y, 0);

            wallMesh.add(septumMesh);

        }

        let generateTheWindow = (x, y) => {

            var windowMesh = new THREE.Mesh(windowGeometry, windowMaterial);

            windowMesh.position.set(x, y, 0);

            wallMesh.add(windowMesh);

        }

        var texture = new THREE.ImageUtils.loadTexture(textureSrc);

        texture.minFilter = THREE.NearestFilter;

        var wallGeometry = new THREE.BoxGeometry(sceneSize \* 2, sceneSize \* 2, 1);

        var wallMaterial = new THREE.MeshPhongMaterial({ map: texture });

        var wallMesh = new THREE.Mesh(wallGeometry, wallMaterial);

        wallMesh.position.set(positionX, positionY, positionZ);

        wallMesh.rotation.x = rotationX;

        wallMesh.rotation.y = rotationY;

        scene.add(wallMesh);

        if (windows) {

            generateTheSeptum(-25, 0, false);

            generateTheWindow(-60, 0);

            generateTheSeptum(-100, -65, true);

            generateTheSeptum(-95, 0, false);

            generateTheWindow(-130, 0);

            generateTheSeptum(-100, 65, true);

            generateTheSeptum(-165, 0, false);

        }

    }

//код для создания двигающейся бочки

let generateBarrel = () => {

        var points = [];

        for (var i = -2; i < 1.5; i = i + 0.1) {

            points.push(new THREE.Vector3(15 + 5 \* Math.exp(-i \* i), 25, 20 \* i));

        }

        var texture = new THREE.ImageUtils.loadTexture('assets/barrelTexture.jpg');

        texture.minFilter = THREE.NearestFilter;

        texture.wrapS = THREE.ClampToEdgeWrapping;

        texture.wrapT = THREE.ClampToEdgeWrapping;

        var barrel\_material = new THREE.MeshPhongMaterial({

            side: THREE.DoubleSide,

            map: texture,

            color: 0xdaa520,

            specular: 0x00b2fc,

            shininess: 50,

            blending: THREE.NormalBlending,

            depthTest: true

        });

        var dno\_geometry = new THREE.CylinderGeometry(29.5, 25, 0, 32);

        var disc1 = new THREE.Mesh(dno\_geometry, barrel\_material);

        disc1.position.set(50, 0, 150);

        var geometry = new THREE.LatheGeometry(points, 32);

        var wall = new THREE.Mesh(geometry, barrel\_material);

        wall.position.set(50, 26, 150);

        wall.rotation.x = Math.PI / 2;

        barrel.add(wall);

        barrel.add(disc1);

        scene.add(barrel);

    }

//код для поверхности из ящиков

let generateBoxesTerrain = () => {

        let terrainSize = 20;

        let boxTexture = new THREE.ImageUtils.loadTexture('assets/barrelTexture.jpg');

        boxTexture.minFilter = THREE.NearestFilter;

        let boxSize = 28;

        let geometry = new THREE.BoxGeometry(boxSize, boxSize, boxSize);

        let material = new THREE.MeshPhongMaterial({

            color: 0x708090,

            side: THREE.DoubleSide,

            depthTest: true,

            map: boxTexture,

        });

        for (let i = 0.3; i <= terrainSize / 4; i++) {

            for (let j = 3; j <= terrainSize / 4; j++) {

                let boxes = new THREE.Mesh(geometry, material);

                boxes.position.set(250, 14, 10);

                boxes.position.x = j \* 70;

                boxes.position.z = i \* 70;

                scene.add(boxes);

            }

        }

    }

//код для анимации огня

 let generateFire = () => {

        const particleCount = 1500;

        const fireSize = 45;

        var particles = new THREE.Geometry();

        var pMaterial = new THREE.PointsMaterial({

            color: 0x808080,

            size: 5

        });

        for (var i = 0; i < particleCount; i++) {

            var pX = Math.trunc(Math.random() \* fireSize) + 28;

            var pY = Math.trunc(Math.random() \* fireSize) + 50;

            var pZ = Math.trunc(Math.random() \* fireSize + 128);

            var particle = new THREE.Vector3(pX, pY, pZ);

            particles.vertices.push(particle);

        }

        var particleSystem = new THREE.Points(

            particles,

            pMaterial

        );

        barrel.add(particleSystem);

        function animate() {

            requestAnimationFrame(animate);

            particles.vertices.forEach(function (particle) {

                particle.y += Math.random() \* 0.3;

                if (particle.y > fireSize + 70) {

                    particle.y = 60;

                }

            });

            particleSystem.geometry.verticesNeedUpdate = true;

        }

        animate();

    }

//код для генерации модели из блендера  
let generateBlenderModel = () => {

        let modelSize = 24;

        function addModelToScene(geometry, materials) {

            var material = new THREE.MeshFaceMaterial(materials);

            var model = new THREE.Mesh(geometry, material);

            model.scale.set(modelSize, modelSize, modelSize);

            model.position.set(54, 25, 270);

            scene.add(model);

        }

        var jsonLoader = new THREE.JSONLoader();

        jsonLoader.load("magic-box.js", addModelToScene);

    }

//код для игры с ловлей капель с помощью ведра

let generateRain = () => {

        const drop\_material = new THREE.MeshBasicMaterial({ color: 0x00A5FF });

        const drop\_circle\_geometry = new THREE.SphereGeometry(3, 32, 32);

        var drop\_part\_geometry = new THREE.CylinderGeometry(0, 3, 15, 32);

        var drop\_part\_mash = new THREE.Mesh(drop\_part\_geometry, drop\_material);

        let dropped;

        const drops = [];

        const drop = new THREE.Mesh(drop\_circle\_geometry, drop\_material);

        drop\_part\_mash.position.set(0, 8, 0)

        drop.add(drop\_part\_mash);

        function addDrop() {

          dropped = false;

          drop.position.x = 140;

          drop.position.y = 400;

          drop.position.z = Math.random() \* sceneSize \* 2;

          scene.add(drop);

          drops.push(drop);

        }

        addDrop();

        function animateDrops() {

          drops.forEach((drop) => {

            if(!dropped){

                // скорость капли

                drop.position.y -= 0.5;

                if ((drop.position.z > bucket.position.z) &&

                    (drop.position.z < bucket.position.z + 60) &&

                    (drop.position.y < 40)) {

                    score += 1;

                    scoreCount.innerHTML = score;

                    dropped = true;

                    scene.remove(drop);

                    setTimeout(addDrop, 2000);

                } else if (drop.position.y < 5) {

                    dropped = true;

                    scene.remove(drop);

                    const dropPieces = [];

                    const pieceCount = 10;

                    const pieceGeometry = new THREE.SphereGeometry(2, 16, 16);

                    const pieceMaterial = drop\_material;

                    for (let i = 0; i < pieceCount; i++) {

                        const piece = new THREE.Mesh(pieceGeometry, pieceMaterial);

                        dropPieces.push(piece);

                        scene.add(piece);

                    }

                    const explosionSpeed = 10;

                    const duration = 0.5;

                    const endTime = performance.now() + duration \* 1000;

                    function animatePieces() {

                        const remainingTime = endTime - performance.now();

                        if (remainingTime <= 0) {

                            dropPieces.forEach((piece) => {

                                scene.remove(piece);

                            });

                        } else {

                            dropPieces.forEach((piece) => {

                                const theta = Math.random() \* Math.PI \* 2;

                                const phi = Math.random() \* Math.PI - Math.PI / 2;

                                const direction = new THREE.Vector3(

                                    Math.sin(theta) \* Math.cos(phi),

                                    Math.sin(phi),

                                    Math.cos(theta) \* Math.cos(phi)

                                );

                                const speed = Math.random() \* explosionSpeed;

                                piece.userData.velocity = direction.multiplyScalar(speed);

                                if(piece.userData.velocity.y > 0){

                                    piece.position.x = piece.userData.velocity.x + 140;

                                    piece.position.y = piece.userData.velocity.y + drop.position.y;

                                    piece.position.z = piece.userData.velocity.z + drop.position.z;

                                }

                            });

                            requestAnimationFrame(animatePieces);

                        }

                    }

                    animatePieces();

                    // скорость генерации капли

                    setTimeout(addDrop, 3000);

                }

            }

          });

        }

        function animate() {

          requestAnimationFrame(animate);

          animateDrops();

        }

        animate();

    };

    let generateBucket = () => {

        var bucket\_material = new THREE.MeshPhongMaterial({

            color: 0xffffff,

        });

        var bucket\_geometry = new THREE.CylinderGeometry(20, 15, 40, 40);

        var bucket\_cover = new THREE.Mesh(bucket\_geometry, bucket\_material);

        bucket\_cover.geometry.computeVertexNormals(false);

        var water\_material = new THREE.MeshBasicMaterial({

            color: 0x00A5FF,

        })

        var water\_geometry = new THREE.CylinderGeometry(18, 5, 39, 39);

        var water = new THREE.Mesh(water\_geometry, water\_material);

        bucket\_cover.position.set(150, 19, 30);

        water.position.set(150, 20, 30);

        bucket.add(water);

        bucket.add(bucket\_cover);

        scene.add(bucket);

    }

//крестики-нолики

let generateTicTacToe = () => {

        const fieldGeometry = new THREE.BoxGeometry(200, 200, 1);

        let fieldTexture = new THREE.ImageUtils.loadTexture('assets/blackboard.jpg');

        fieldTexture.minFilter = THREE.NearestFilter;

        const fieldMaterial = new THREE.MeshBasicMaterial({ map: fieldTexture });

        field = new THREE.Mesh(fieldGeometry, fieldMaterial);

        const horizontalGeometry = new THREE.BoxGeometry(150, 1, 1);

        const verticalGeometry = new THREE.BoxGeometry(1, 150, 1);

        const firstLine = new THREE.Mesh(horizontalGeometry, deskMaterial);

        const secondLine = new THREE.Mesh(horizontalGeometry, deskMaterial);

        const thirdLine = new THREE.Mesh(verticalGeometry, deskMaterial);

        const fourthLine = new THREE.Mesh(verticalGeometry, deskMaterial);

        firstLine.position.set(0, 25, 1);

        secondLine.position.set(0, -25, 1);

        thirdLine.position.set(25, 0, 1);

        fourthLine.position.set(-25, 0, 1);

        field.add(firstLine, secondLine, thirdLine, fourthLine);

        field.position.set(290, 150, 1);

        scene.add(field);

    }

Полученные результаты (скриншоты)

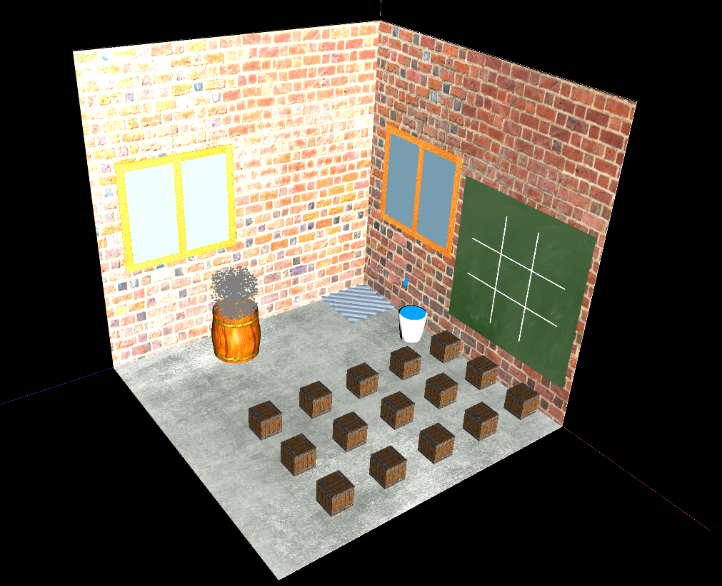


Рисунок 1. Сцена целиком



Рисунок 2. Анимированная бочка

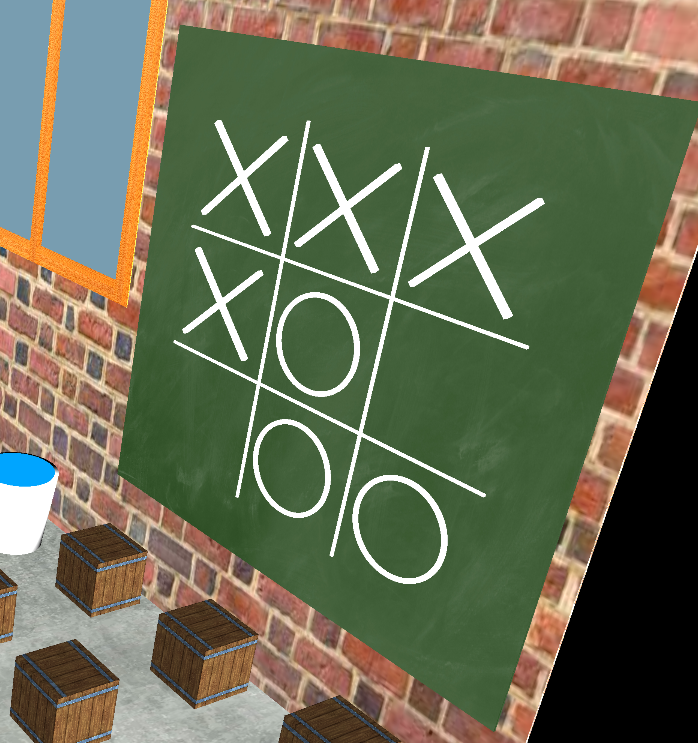


Рисунок 3. Интерактивные крестики-нолики



Рисунок 4. Передвигаемое ведро и упавшая капля воды



Рисунок 5. Поверхность из ящиков

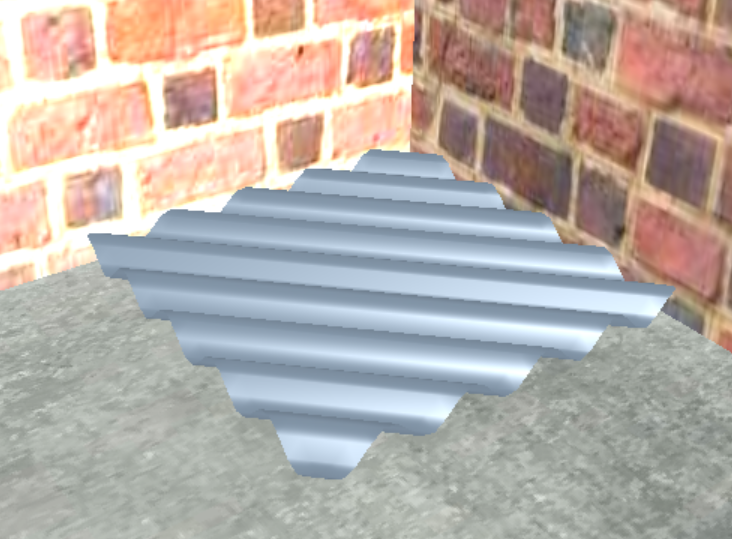


Рисунок 6. Металлический шифер

Программная документация

Руководство пользователя:

* 1, 2, 3, 4, 5, 6, 7, 8, 9 для игры в «крестики-нолики»
* A, W, S, D для передвижения бочки
* Z, X для передвижения ведра

Используемые технологии и инструменты:

1. Three.js — это библиотека JavaScript для создания 3D-графики в веб-браузере.

2. Blender — это свободное и открытое программное обеспечение для моделирования 3D-графики, которое использовалось для создания и импорта 3D-моделей объектов в приложении.

3. HTML, CSS и JavaScript — это основные языки программирования, используемые в веб-разработке. Мы использовали их для создания пользовательского интерфейса, анимации и взаимодействия с пользователем.

4. Git — это распределенная система контроля версий, которую мы использовали для управления и отслеживания изменений в коде приложения.

5. GitHub — это онлайн-платформа для хранения и совместной работы над проектами с использованием системы контроля версий Git. Мы использовали ее для хранения кода приложения.

6. Visual Studio Code — это бесплатный редактор кода для кроссплатформенной разработки веб-приложений.