**Липецкий государственный технический университет**  
Факультет автоматизации и информатики  
Кафедра автоматизированных систем управления

Лабораторная работа №3

по предмету «Организация графических систем и систем мультимедиа»

Создание 3D сцены с использованием WebGL

Максимов А.В.

Студент

Группа М-АС-21

Кургасов В.В.

Руководитель

Доцент

Липецк 2022 г.

Задание кафедры

Создать 3D-сцену (не менее 3-х объектов) на WebGL для отображения в браузере.

Содержание

[Теоретическая часть 4](#_Toc103097692)

[Ход работы 4](#_Toc103097693)

[Вывод 7](#_Toc103097694)

[Приложение А 8](#_Toc103097695)

[Приложение Б 15](#_Toc103097696)

Теоретическая часть

WebGL (Web Graphics Library) - программная библиотека для языка JavaScript предназначенная для визуализации интерактивной трёхмерной графики и двухмерной графики в пределах совместимости веб-браузера без использования плагинов. WebGL приносит в веб трёхмерную графику, вводя API, который построен на основе OpenGL ES 2.0, что позволяет его использовать в элементах canvas HTML5.

Для разработки был выбран фреймворк Vue.js и библиотека Three.js.

Three.js – это библиотека JavaScript, содержащая набор готовых классов для создания и отображения интерактивной 3D графики в WebGL.

Three.js предлагает декларативный синтаксис, и абстрагирует от «головных болей», связанных с отображением 3D графики в браузере.

Моделирование графики с использованием Three.js можно сравнить со съемочной площадкой, так как у нас есть возможность оперировать такими понятиями как сцена, свет, камера, объекты и их материалы.  
Три, так называемых, кита Three.js включают в себя:

* Scene — своеобразная платформа, где размещаются все объекты, которые мы создаем;
* Camera — по сути — это “глаз”, который будет направлен на сцену. Камера снимает и отображает объекты, которые расположены на сцене;
* Renderer — визуализатор, который позволяет показывать сцену, снятую камерой.

Ход работы

Приложение отображает сцену с тремя 3D моделями автомобилей, при клике на которые открывается страница с характеристиками автомобилей.

На рисунке 1 представлена сцена с 3D моделями автомобилей



Рисунок 1 – Сцена с 3D моделями

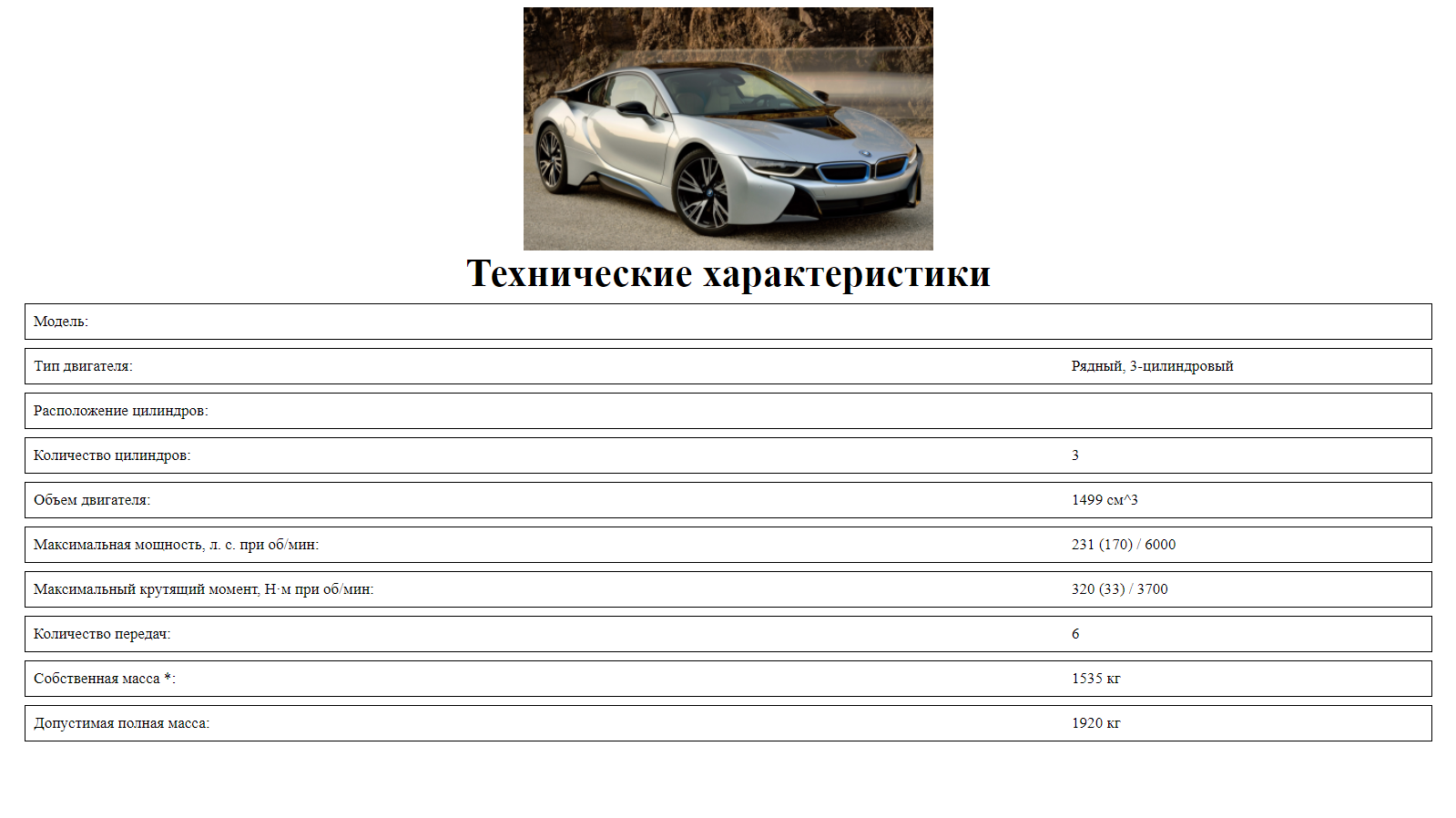


Рисунок 2 – Страница с техническими характеристиками автомобиля

Исходный код данных страницы представлен в приложениях А и Б.

Чтобы иметь возможность динамически изменять положение камеры, к компоненту Renderer был добавлен атрибут orbit-ctrl.

Компонент GltfModel отвечает за загрузку моделей. Атрибут position этого компонента задаёт позицию объекта на сцене по x, y и z координатам. Rotation отвечает за поворот по координатам. В атрибут @click задаётся функция, срабатывающая при клике на модель.

Для создания docker образа приложения добавим Dockerfile в проект. На рисунке 2 представлено содержание этого файла.

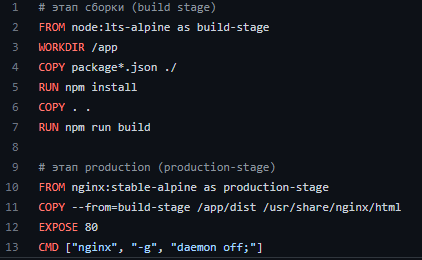


Рисунок 3 – Содержание Dockerfile

Загрузим проект на сервер, соберём и запустим.

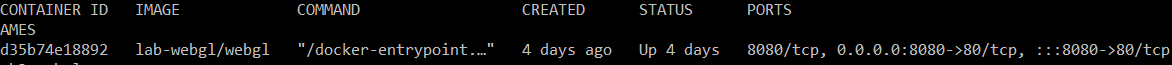


Рисунок 4 – Результат запуска докер контейнера на сервере

Вывод

В ходе выполнения лабораторной работы была разработана программа, отображающая 3D-сцену с тремя объектами на WebGL для отображения в браузере. Приложение реализовано с использованием фреймворка Vue.js и библиотеки three.js.

С помощью инструмента Docker приложение было упаковано в докер контейнер и запущено на сервере.

Приложение А

Код главной страницы со сценой

<template>

<div class="settings">

<div>

<h5>Model</h5>

<label for="model-x">x</label>

<input type="range" id="model-x" v-model="modelX" min="-1000" max="1000">

<label for="model-y">y</label>

<input type="range" id="model-y" v-model="modelY" min="-1000" max="1000">

<label for="model-z">z</label>

<input type="range" id="model-z" v-model="modelZ" min="-1000" max="1000">

</div>

<div>

<h5>Camera</h5>

<label for="camera-x">x</label>

<input type="range" id="camera-x" min="-1000" max="1000">

<label for="camera-y">y</label>

<input type="range" id="camera-y" min="-1000" max="1000">

<label for="camera-z">z</label>

<input type="range" id="camera-z" min="-1000" max="1000">

</div>

</div>

<Renderer ref="renderer" resize="window" orbit-ctrl :pointer="{ intersectRecursive: true }">

<Camera :position="{x: 0, y: 345 ,z: 1000 }" :far="6000"/>

<Scene ref="scene" background="#0f0f0f">

<HemisphereLight />

<Group>

<GltfModel src="./audiQ7/scene.gltf"

@click="carOne"

ref="car1"

:position="{x: 5, y: 5, z: 5 }"

:rotation="{ y: rotY }"

/>

<GltfModel src="./audi\_r8/scene.gltf"

@click="carTwo"

ref="car2"

:position="{x: 504, y: 62, z: -1000 }"

:rotation="{ y: rotY }"

/>

<GltfModel src="./bmw\_i8/scene.gltf"

@click="carThree"

ref="car3"

:position="{x: -894, y: 115 , z: -1000 }"

:rotation="{ y: rotY }"

/>

</Group>

</Scene>

</Renderer>

</template>

<script>

import { defineComponent, onMounted, ref } from 'vue'

import {

Camera, Renderer,

Scene, GltfModel, HemisphereLight, PointLight, Group

} from 'troisjs'

import { useRouter } from 'vue-router'

export default defineComponent({

name: 'Canvas',

components: { Camera, HemisphereLight, Renderer, Scene, GltfModel, PointLight, Group },

setup () {

const router = useRouter()

const renderer = ref()

const scene = ref()

const rotY = ref(0)

const modelX = ref(5)

const modelY = ref(5)

const modelZ = ref(5)

const carAudiQ7 = ref({

img: 'audiQ7.jpg',

model: '45 TDI quattro',

dvsType: '45 TDI',

cylArrangement: 'V',

numCyl: '6',

eCapacity: '2967 см^3',

maxPower: '249 / 3000-4500',

maxTorque: '600 / 1500-2750',

numTransmission: '8',

deadweight: '1980 кг',

totalMass: '2750 кг'

})

const carAudiR8 = ref({

img: 'audi\_r8.jpg',

model: '5.2 FSI quattro S tronic',

dvsType: '',

cylArrangement: 'V',

numCyl: '10',

eCapacity: '2967 см^3',

maxPower: '540 (397) / 7800',

maxTorque: '540 / 6500',

numTransmission: '7',

deadweight: '1595 кг',

totalMass: '1895 кг'

})

const carBmwI8 = ref({

img: 'bmw\_i8.jpg',

model: '',

dvsType: 'Рядный, 3-цилиндровый',

cylArrangement: '',

numCyl: '3',

eCapacity: '1499 см^3',

maxPower: '231 (170) / 6000',

maxTorque: '320 (33) / 3700',

numTransmission: '6',

deadweight: '1535 кг',

totalMass: '1920 кг'

})

const audiQ7 = JSON.stringify(carAudiQ7.value)

const bmwI8 = JSON.stringify(carBmwI8.value)

const audiR8 = JSON.stringify(carAudiR8.value)

const carOne = () => {

router.push({

path: '/car',

name: 'Car',

query: { carName: 'audiQ7' },

params: {

car: audiQ7

}

})

}

const carTwo = () => {

router.push({

path: '/car',

name: 'Car',

query: { carName: 'audiR8' },

params: {

car: audiR8

}

})

}

const carThree = () => {

router.push({

path: '/car',

name: 'Car',

query: { carName: 'bmwI8' },

params: {

car: bmwI8

}

})

}

const rotationObject = () => {

rotY.value -= 0.005

}

onMounted(() => {

renderer.value.onBeforeRender(() => {

rotationObject()

})

})

return {

renderer,

scene,

rotY,

modelX,

modelY,

modelZ,

audiQ7,

audiR8,

carOne,

carTwo,

carThree

}

}

})

</script>

<style scoped>

.settings {

display:none;

flex-direction: row;

width: fit-content;

background-color: #cec8c8;

}

.settings > div {

display:flex;

flex-direction: column;

margin-left: 20px;

}

</style>

Приложение Б

<template>

<img :src="require(`../assets/${carData.img}`)" alt="car">

<h2>Технические характеристики</h2>

<ul>

<li>

<p>Модель:</p>

<p>{{carData.model}}</p>

</li>

<li>

<p>Тип двигателя:</p>

<p>{{carData.dvsType}}</p>

</li>

<li>

<p>Расположение цилиндров:</p>

<p>{{carData.cylArrangement}}</p>

</li>

<li>

<p>Количество цилиндров:</p>

<p>{{carData.numCyl}}</p>

</li>

<li>

<p>Объем двигателя:</p>

<p>{{carData.eCapacity}}</p>

</li>

<li>

<p>Максимальная мощность, л. с. при об/мин:</p>

<p>{{carData.maxPower}}</p>

</li>

<li>

<p>Максимальный крутящий момент, Н·м при об/мин:</p>

<p>{{carData.maxTorque}}</p>

</li>

<li>

<p>Количество передач:</p>

<p>{{carData.numTransmission}}</p>

</li>

<li>

<p>Собственная масса \*:</p>

<p>{{ carData.deadweight }}</p>

</li>

<li>

<p>Допустимая полная масса:</p>

<p>{{carData.totalMass}}</p>

</li>

</ul>

</template>

<script>

import { defineComponent, ref } from 'vue'

export default defineComponent({

name: 'Car',

props: {

car: {

type: String

}

},

setup (props, \_) {

// eslint-disable-next-line vue/no-setup-props-destructure

const carData = ref(JSON.parse(props.car))

return { carData }

}

})

</script>

<style scoped>

img {

display: flex;

margin: 10px auto 0 auto;

width: 500px;

height: auto;

alignment: center;

}

h2 {

text-align: center;

font-size: 48px;

}

ul {

display: flex;

flex-direction: column;

}

ul > li{

display: flex;

flex-direction: row;

justify-content: space-between;

margin: 10px 30px 0 30px;

border: 1px solid black;

}

ul > li:last-child {

margin-bottom: 10px;

}

ul > li > p{

width: 25%;

font-size: 19px;

text-align: start;

padding: 10px;

}

</style>