**Министерство науки и высшего образования Российской Федерации**

**федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

**(Университет ИТМО)**

**Факультет** Программной инженерии и компьютерной техники (ФПИ и КТ)

**Образовательная программа:** Системное и прикладное программное обеспечение

**Направление подготовки (специальность)**: 09.03.04, Программная инженерия

О Т Ч Е Т

*о производственной, технологической практике*

Тема задания: *Расширение функциональности среды просмотра mtlx-формата в 3D в браузере*

Обучающийся: *Кульбако Артемий Юрьевич, P34112*

Руководитель практики от профильной организации: *Кравченко Андрей Сергеевич, Senior Programmer в ООО "Люксофт Профешнл"*

Руководитель практики от университета: *Маркина Татьяна Анатольевна, доцент*

Дата: 07.03.2022

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc99804110)

[1. ВВЕДЕНИЕ 3](#_Toc99804111)

[2. ХАРАКТЕРИСТИКА ОРГАНИЗАЦИИ 5](#_Toc99804112)

[3. ОПИСАНИЕ ВЫПОЛНЕНИЯ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ 6](#_Toc99804113)

[3.1 Исправить существующие ошибки и недочёты 6](#_Toc99804114)

[3.3 Сделать окно параметров плавающим 9](#_Toc99804115)

[3.4 Заменить проблемные меши на меши с правильной UV 11](#_Toc99804116)

[3.5 Добавить поддержку различных эффектов постобработки 13](#_Toc99804117)

[3.6 Имплементировать систему уведомлений с основного фрондента matlib.gpuopen 14](#_Toc99804118)

[3.7 Добавить анимацию загрузки 17](#_Toc99804119)

[3.8 Реализовать управление направленным освещением 20](#_Toc99804120)

[3.9 Мобильная версия сайта фронтенда matlib.gpuopen 21](#_Toc99804121)

[3.10 Исследовать поддержку WebGPU вместо WebGL 24](#_Toc99804122)

[3.11 Добавить поддержку nodedef-нодов если есть поддержка ядром 25](#_Toc99804123)

[3.12 Оформление отчётности 27](#_Toc99804124)

[4. ЗАКЛЮЧЕНИЕ 28](#_Toc99804125)

[5. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29](#_Toc99804126)

# 1. ВВЕДЕНИЕ

Производственная практика – заключительный и важнейший этап образовательного процесса, направленный на проверку и закрепление компетенций обучающегося, полученных в процессе академического обучение и учебной практики путём работы над настоящим проектом в условиях, не отличающихся от обычного трудового распорядка компании, принимающей практику.

Целью производственной практики является демонстрация учащимся того, что он способен выполнять работу в рамках своей специальности. Проверка навыков осуществляется через выполнение индивидуального задания (*Таблица 1*).

Таблица 1 - Индивидуальное задание

|  |  |  |
| --- | --- | --- |
| **Порядковый № этапа** | **Наименование этапа** | **Задание этапа** |
| 1 | Исправить существующие ошибки и недочёты | Ознакомиться с выводами команды QA и 3D-artist, обсудить текущую и возможные реализации (при необходимости разработать прототипы). Внести изменения согласно принятым в ходе дискуссий решениям. |
| 2 | Реализовать поддержку алгоритм(а/ов) сглаживания | Устранить эффект "лесенок" на краях мешей. Для этого необходимо сравнить поддерживаемые библиотекой `three.js` существующие алгоритмы сглаживания. Выбрать наилучший по критериям качества-производительности и реализовать его в viewer-е. |
| 3 | Сделать окно параметров материала плавающим | Реализовать возможность свободно перемещать окно параметров материала над основным интерфейсом программы. |
| 4 | Заменить проблемные меши на меши с правильной UV | Заменить меши с неправильной UV-развёрткой (шейдербол, цилиндр). Найти бесплатные на бирже 3D-моделей (обращать внимание на ограничения, накладываемые лицензией), сгенерировать программно методами three.js или самостоятельно создать в 3D-редакторе. |
| 5 | Добавить поддержку различных эффектов постобработки | Изучить поддерживаемые three.js эффекты постобработки, выбрать самые интересные/полезные и создать меню, где можно будет включать и отключать их. |
| 6 | Имплементировать систему уведомлений с основного фрондента matlib.gpuopen | Изучить возможность проброса данных из `iframe` в родительский контейнер. Перехватывать все ошибки viewer-а и отправлять их в систему уведомлений основного фронтенда matlib.gpuopen. |
| 7 | Добавить анимацию загрузки | Разработать прототипы различных анимаций для действий загрузки и смены меша/окружений. Предоставить варианты команде 3D-artist. Лучший из вариантов добавить в viewer. |
| 8 | Реализовать управление направленным освещением | Создать меню, которые позволит управлять направленным освещением сцены по трём параметрам: направлению, цвету и интенсивностью. |
| 9 | Мобильная версия сайта фронтенда matlib.gpuopen | Починить (или реализовать альтернативную) разметку для корректной работы на мобильных устройства. |
| 10 | Исследовать поддержку WebGPU вместо WebGL | Исследовать текущий статус API WebGPU, его поддержку современными браузерами и библиотекой Three.js. |
| 11 | Добавить поддержку nodedef-нодов если есть поддержка ядром | Добавить поддержку nodedef-нодов графа MaterialX взамен nodegraph-нодов, что устарели в версии 1.38.3, если ядро проекта, отвечающее за генерацию шейдеров, уже их поддерживает. |
| 12 | Оформление отчётности | Оформить необходимые документы и отчёты о процессе и результатах прохождения производственной практики. |

# 2. ХАРАКТЕРИСТИКА ОРГАНИЗАЦИИ

ООО «Люксофт Профешнл» (Luxoft, a DXC Technology Company) – общество с ограниченной ответственностью, глобальная компания, оказывающая услуги по разработке программного обеспечения по аутсорсинговой модели. Luxoft создаёт технологические решения в сферах автомобилестроения, финансы, путешествия, биотехнологии, медиа и коммуникации. Офисы компании расположены по всеми миру, главный в Цуг, Швейцария.

Санкт-Петербургское подразделение Luxoft преимущественно занимается разработками для автомобилестроения и компьютерной графикой.

Схема структуры управления (*Рисунок 1*). Жизненный цикл каждого продукта, от заключения контракта главой отдела, проходит по цепочке вниз, к людям, которые этот продукт будут разрабатывать, а потом снова вверх, в процессе верификации и приёма. Благодаря такой структуре, для каждого продукта/проекта параллельного команде разработки существует своя команда тестирования, рекрутинга и рекламы, которая знает, как наиболее эффективно осуществлять свою деятельность для конкретного продукта/проекта.

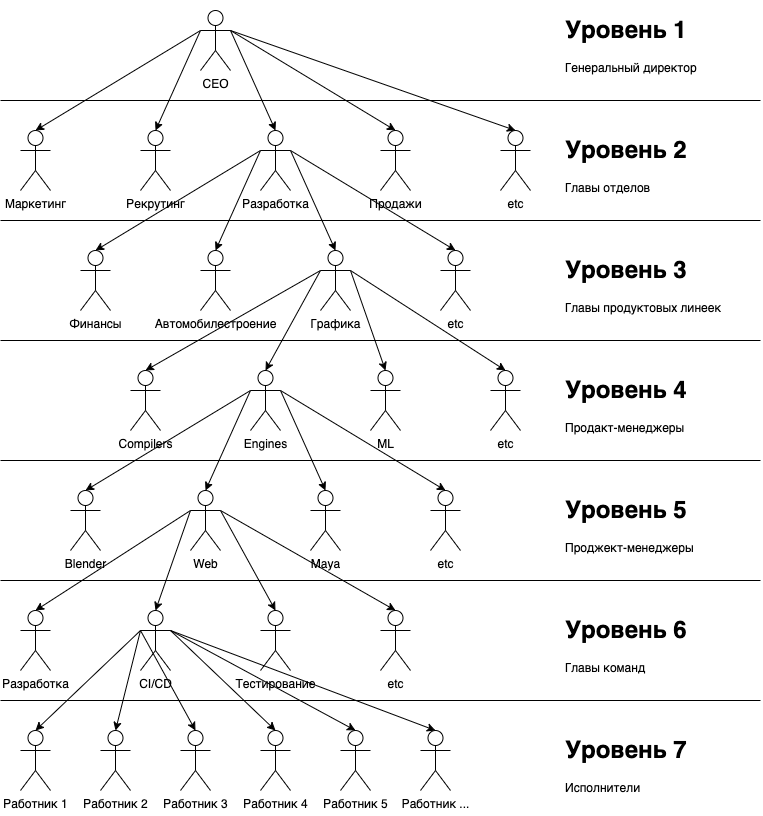


Рисунок 1

# 3. ОПИСАНИЕ ВЫПОЛНЕНИЯ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

## 3.1 Исправить существующие ошибки и недочёты

После того, как команды QA и 3d-artist провели детальный обзор результатов моей работы, проведённой в рамках учебной практики, были обнаружены две проблемы.

Первая – несоответствие стиля полосы прокрутки стилю сайта. Я просто забыл про неё. Проблема решилась подключением файла стилей в проект (*Рисунок 2*, *Таблица 2*).



Рисунок 2

Таблица 2

|  |
| --- |
| \* {  scrollbar-width: thin;  scrollbar-color: map-get($grey, 'base') map-get($shades, 'transparent');  }  \*::-webkit-scrollbar {  width: 8px;  height: 8px;  }  \*::-webkit-scrollbar-track {  border-radius: 10px;  }  \*::-webkit-scrollbar-thumb {  border-radius: 10px;  background-color: map-get($grey, 'base');  } |

Вторая – артефакты на краях мешей – яркие пиксели, образующие эффект «лесенки», появившиеся из-за четырёхкратно возросшего разрешения рендера (*Рисунок 3*).

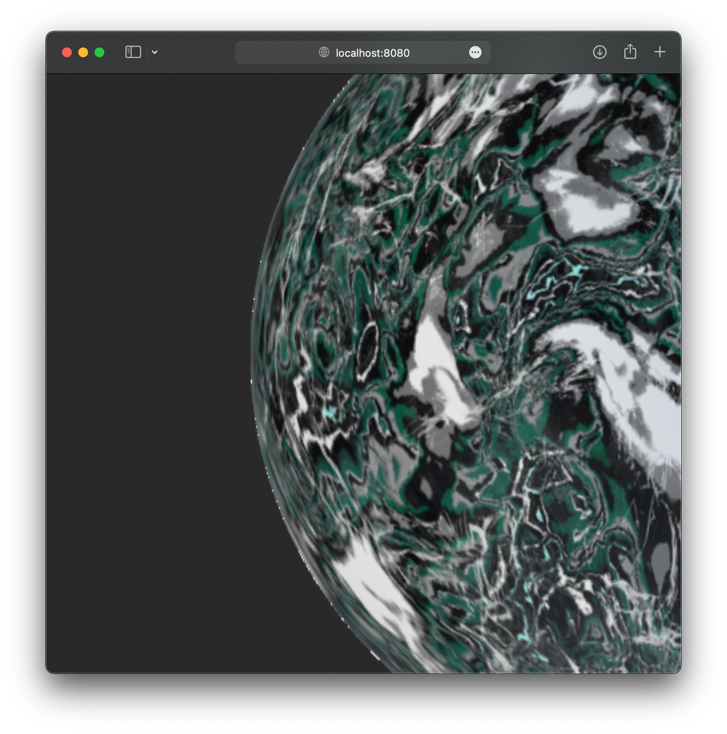


Рисунок 3

Мною было найдено три варианта решения этой проблемы: сделать разрешение рендера низким, каким оно было изначально, но это бы значительно ухудшило качество изображения; реализовать эффект глубины резкости: держать в фокусе камеры только центральную часть изображения, таким образом, края мешей становились нечёткими, цвета ярких пикселей просто замещались другими, но это порождало ещё один артефакт: та часть сцены, которая не содержала меш, становилась полностью чёрной (*Рисунок 4*), последний вариант, который, к тому же, был частью плана производственной практики – реализация алгоритма сглаживания: усреднение цветов пикселей на краях объектов. По этому пути и было решено пойти.

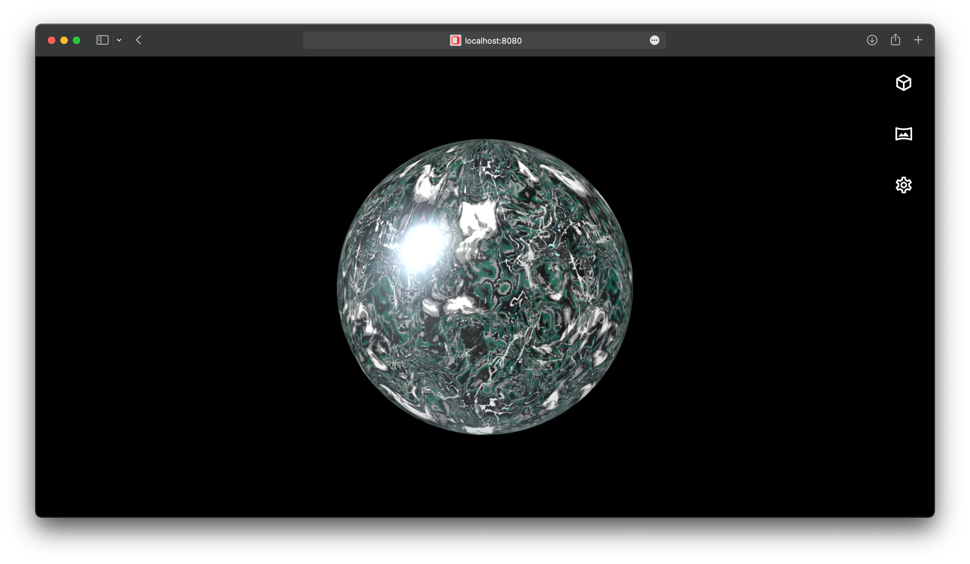


Рисунок 4

**3.2 Реализовать поддержку алгоритм(а/ов) сглаживания**

Для наилучшего результата необходимо реализовать несколько известных алгоритмов сглаживания, а после сравнивать результат по качеству/производительности и выбрать наилучший, пример простейшего алгоритма сглаживания (*Таблица 3*). Тем не менее, библиотека three.js имеет несколько готовых алгоритмов, сложность заключается в внедрении его в пайплайн рендера, так как результат работы (рендер) некоторых алгоритмов зависит от прошлых рендеров, также необходимо подобрать наилучшие параметры для каждого из них. На изображении ниже примеры работы четырёх сглаживаний и их производительности TAA, SSAA, SMAA, FXAA (*Рисунок 5*) (разница видна в основном в динамике).

Таблица 3

|  |
| --- |
| const pixelShift = (srcCanvas, shiftCanvas, xOffset, yOffset) => {  const shiftXY = { xS: 0, yS: 0 }  const shiftCanvasWidth = shiftCanvas.width  const shiftCanvasHeight = shiftCanvas.height  for ( let x = 0; x < shiftCanvasWidth; x ++)  for ( let y = 0; y < shiftCanvasHeight; y++) {  readShift(shiftCanvas, x, y, shiftXY)  writePixel(srcCanvas, xOffset + x , yOffset + y, readPixelColor(srcCanvas, xOffset + shiftXY.xS, yOffset + shiftXY.yS))  }  } |

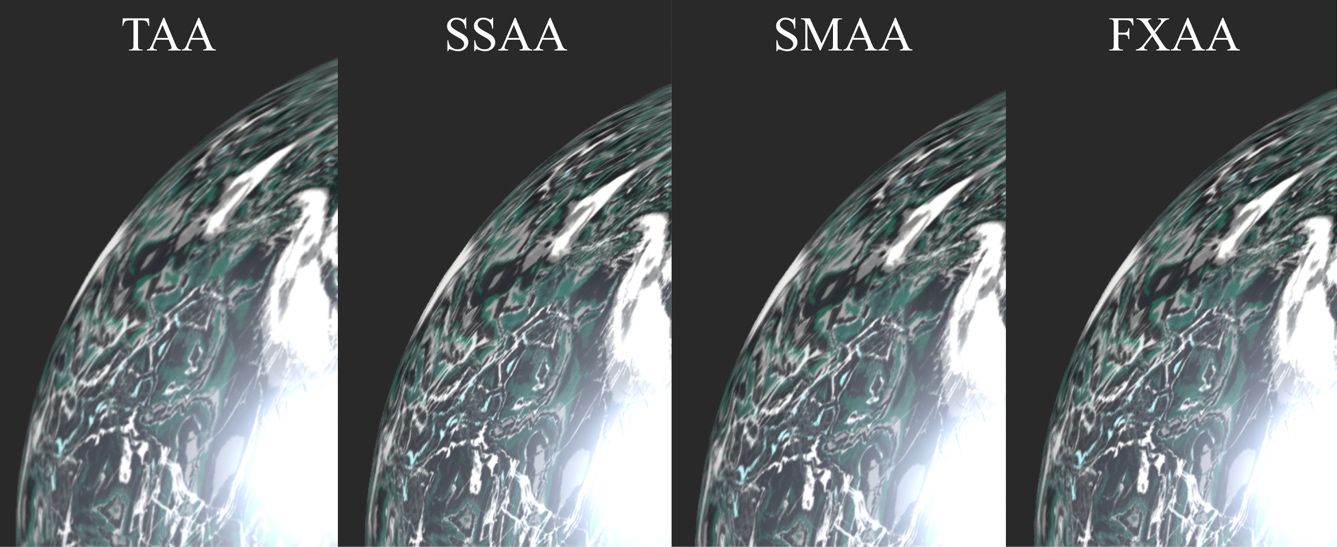


Рисунок 5

SSAA и SMAA обеспечивали наилучшее качество рендера, но ввиду повышения чёткости, яркие пиксели на краях мешей оставались, хоть их количество и уменьшалось. TAA и FXAA немного «мылят» изображение, что является недостатком этих алгоритмов в общем случае, но в нашей ситуации позволило избавиться от артефактов. Между TAA и FXAA предпочтение было отдано последнему, так как рендер следующего кадра не зависит от предыдущих, что позволяет нам рендерить по одному кадру на изменение материала и экономить вычислительные ресурсы пользователя.

## 3.3 Сделать окно параметров плавающим

Окно параметров материала занимает значительную часть всего viewer-а, что негативно сказывается на опыте использования (*Рисунок 6*). Команда 3d-artist предложила разрешить свободно перемещать его по экрану. Задача нетривиальная, так как теги html, Vue или компоненты Vuetify подобной функциональности по умолчанию не представляют.

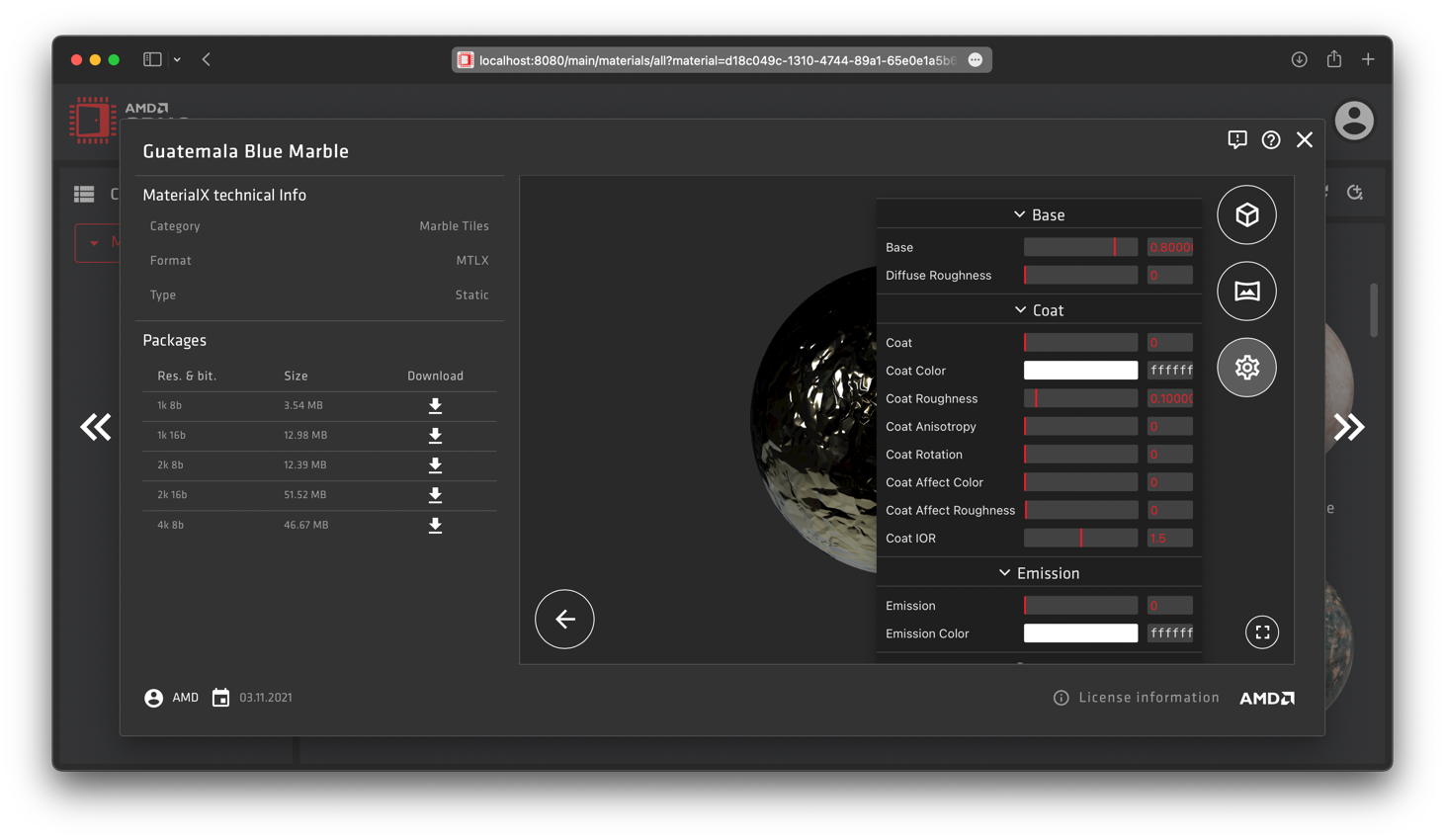


Рисунок 6

В первую очередь я принялся искать готовое решение в репозитории пакетов для JavaScript – npm и там мною был найден подходящий компонент. Следующим шагом было необходимо внедрить уничтожение/создание этого компонента при нажатии на кнопку «Property Editor», прикрепление его к контейнеру, а также сохранение параметров материалов при пересоздании компонента, чтобы пользователи не теряли свои настройки, когда изменяют меш и/или окружение (*Таблица 4*). В конце необходимо было внести небольшие правки по дизайну, предложенные другими членами команды на общем обсуждении. Новый вариант - *Рисунок 7*.

Таблица 4

|  |
| --- |
| watch: {  //...  isParamsMenuVisible: {  immediate: true,  handler() { if (this.isParamsMenuVisible === true) document.getElementById("paramsHolder").appendChild(this.propEditor.domElement) }  }  //...  }  handleMeshOrEnvChange() {  this.swapScene(this.propEditor?.save())  this.propEditor?.destroy()  this.propEditor = new GUI({ autoPlace: false })  this.propEditor.domElement.childNodes[0].remove()  }  const fillPropertyEditor = (material, uniforms, propEditor, callback, state) => {  const PARAM\_TYPE = {  ZERO\_TO\_ONE: Symbol('ZERO\_TO\_ONE'),  COLOR: Symbol('COLOR'),  NUMERIC: Symbol('NUMERIC')  }  const addUniformParam = (folder, paramType, uniformName, customParamName, from, to, step) => {  const p = uniforms[uniformName]  if (p) {  const paramName = customParamName ?? uniformName.replaceAll('\_', ' ').split(' ').map(capitalize).join(' ')  switch (paramType) {  case PARAM\_TYPE.ZERO\_TO\_ONE:  folder.add(p, 'value', 0, 1).step(0.0001).name(paramName).onChange( () => callback() )  break  case PARAM\_TYPE.COLOR:  folder.addColor(new ColorGUIController(uniforms, uniformName), 'value').name(paramName).onChange( () => callback() )  break  case PARAM\_TYPE.NUMERIC:  folder.add(uniforms[uniformName], 'value', from, to).step(step).name(paramName).onChange( () => callback() )  break  }  }  }  const base = propEditor.addFolder('Base')  addUniformParam(base, PARAM\_TYPE.ZERO\_TO\_ONE, 'base')  //...  propEditor.add({ reset() { propEditor.reset() } }, 'reset').name('Reset')  if (state) propEditor.load(state)  } |

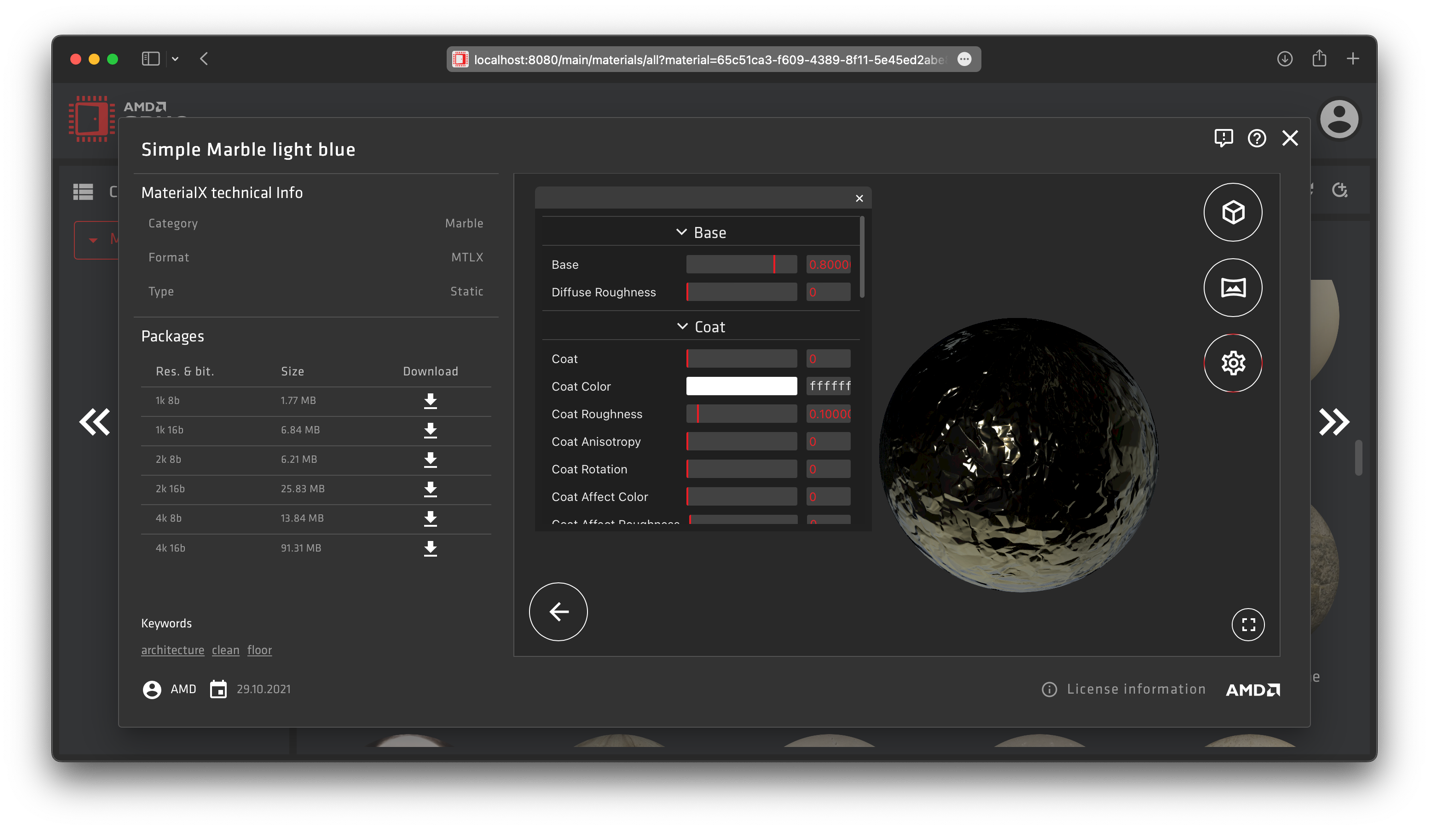


Рисунок 7

## 3.4 Заменить проблемные меши на меши с правильной UV

UV-развёртка – это набор правил, описывающий каким образов текстура должна применятся к определённому мешу, в общем случае хранится в одном файле вместе с мешем, создаётся в 3D-редакторе.

Необходимо было самому создать меши с правильной развёрткой или найти бесплатные в интернете, лицензия которых позволяет использовать их в любых целях. Библиотека Three.js поддерживает два типа создания мешей: программный (написание кода) и объектный (импорт файла, созданного в 3D-редакторе). Также стоит учитывать, что чем больше полигонов у меша (граней геометрической фигуры), тем больше ресурсов расходуется на рендеринг, и тем сложнее написать его код. Тем не менее, при одинаковом количестве полигонов, программный метод отработает быстрее, так как js-код весит меньше, чем бинарный файл и его не надо загружать ассинхронно.

Результатом работы стало хранилище мешей, которое позволяет вызывающему коду получать меш по его имени, реализация создания же для него скрыта. Простые меши я стал создавать программно, а сложные оставил в виде файла. На программно-создаваемых мешах стандартная UV-развёртка оказалось правильной. На сложных мешах проблем с ней не было. Реализация хранилища (*Таблица 5*) и пример меша с неправильной и правильной развёртками (*Рисунок 8*).

Таблица 5

|  |
| --- |
| import { GLTFLoader } from "three/examples/jsm/loaders/GLTFLoader"  import { DRACOLoader } from "three/examples/jsm/loaders/DRACOLoader"  import { CylinderGeometry, Mesh, PlaneGeometry } from "three"  const GLTF\_LOADER = new GLTFLoader()  GLTF\_LOADER.setDRACOLoader(new DRACOLoader())  const BASE\_MESHES\_PATH = 'public/meshes/'  const loadFromFS = async name =>  new Promise(resolve => GLTF\_LOADER.load(`${BASE\_MESHES\_PATH}${name}.glb`, resolve))  .then( mesh => mesh.scene )  const getPreviewPaths = name => ({  previewPath: `${BASE\_MESHES\_PATH}${name}\_preview.png`,  previewPathSelected: `${BASE\_MESHES\_PATH}${name}\_preview\_selected.png`  })  const MESHES\_FACTORY = [  {  name: 'Sphere',  builder: async () => loadFromFS('sphere'),  ...getPreviewPaths('sphere')  },  {  name: 'Shaderball',  builder: () => loadFromFS('shaderball'),  ...getPreviewPaths('shaderball')  },  {  name: 'Cylinder',  builder: () => new Mesh(new CylinderGeometry(0.4, 0.4, 1.68, 60)),  ...getPreviewPaths('cylinder')  },  {  name: 'Cloth',  builder: () => loadFromFS('cloth'),  ...getPreviewPaths('cloth')  },  {  name: 'Plane',  builder: () => new Mesh(new PlaneGeometry(5, 5)),  ...getPreviewPaths('plane')  }  ]  export { MESHES\_FACTORY } |



Рисунок 8

## 3.5 Добавить поддержку различных эффектов постобработки

Необходимо добавить различные эффекты постобработки, выбор эффектов по моему усмотрению.

Сначала я изучил список эффектов, которые поддерживает Three.js. Основная проблема заключается в том, что не все доступные эффекты будут работать ввиду того, что часть из них должна взаимодействовать с шейдерами, отвечающими за просчёт света, а в нашем случае этим занимается инкапсулированный код, подключаемый в виде отдельного WebAssembly модуля. Другая часть эффектов просто бессмысленна, к примеру, глубина резкости, так как за мешем мы не отображаем фон (в расфокусе будет «ничего»). Мне нужно было выбирать эффекты, которые не относятся к двум перечисленным группам.

В итоге, я реализовал несколько эффектов: «глитч», отображение полигональной сетки, SSAO – улучшенные тени, имитация зернистости плёнки (пример *Рисунок 9*). Реализации одного из эффектов, отображение сетки (*Таблица 6*).

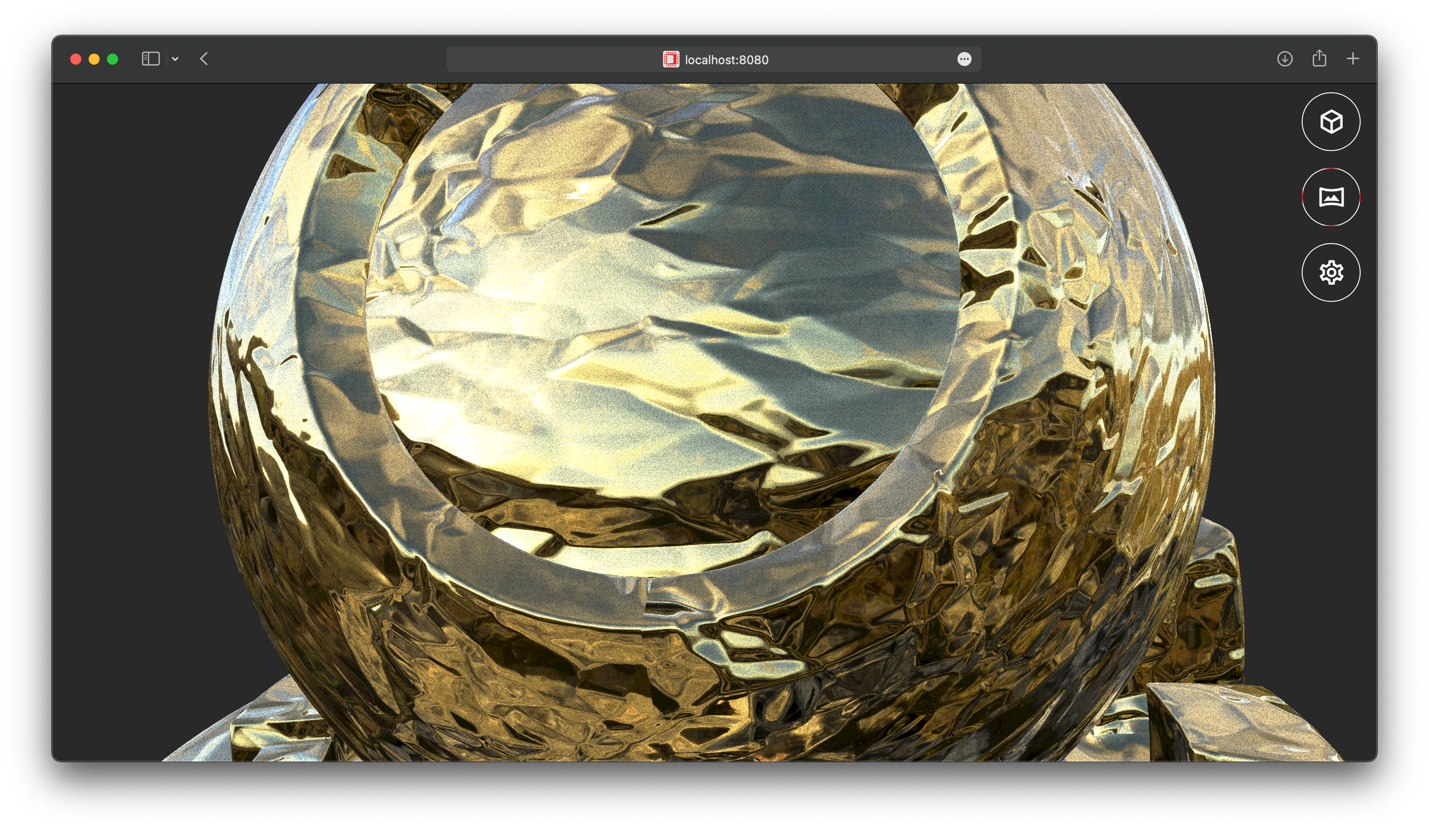


Рисунок 9

Таблица 6

|  |
| --- |
| import { Group, Object3D, LineSegments, WireframeGeometry, LineBasicMaterial } from "three"  import 'core-js/actual/array/group-by'  [Group, Object3D].forEach( it =>  it.prototype.wireframe = function(color = 'black') {  return this.clone(true).apply( s =>  s.traverse( child => {  if (child.isMesh) {  const wireframe = new LineSegments(  new WireframeGeometry(child.geometry),  new LineBasicMaterial({ color: color })  )  child.add(wireframe)  child.parent?.attach(wireframe)  child.parent?.remove(child)  }  })  )  }) |

## 3.6 Имплементировать систему уведомлений с основного фрондента matlib.gpuopen

В случае возникновения ошибок необходимо выводить уведомление с ними, стиль и поведение уведомлений должны соответствовать основному фронтенду сайта (пример *Рисунок 10*).

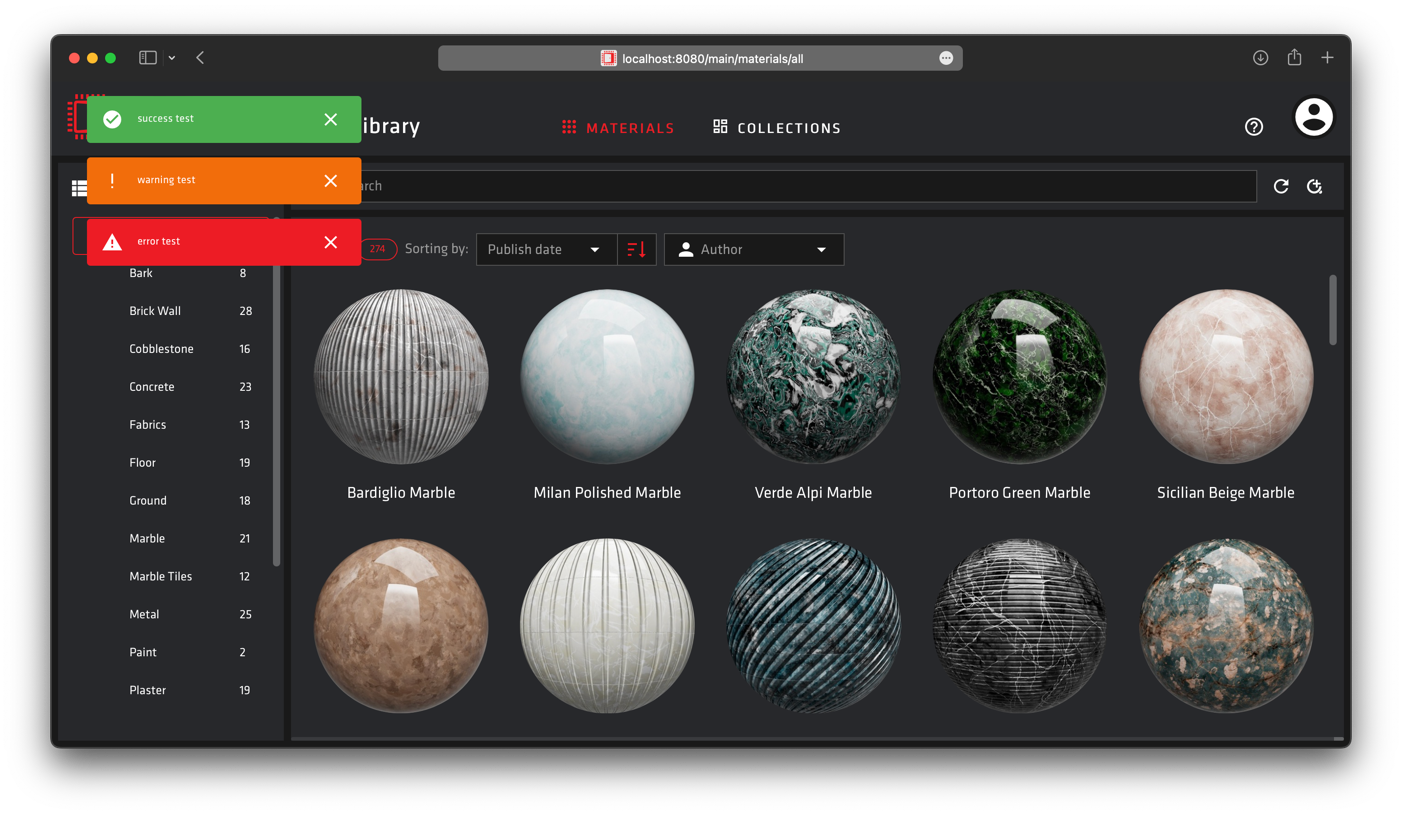


Рисунок 10

Первый, и достаточно очевидный вариант, просто скопировать весь код модуля уведомлений, но у такого подхода есть фатальный недостаток: ввиду того, что viewer – отдельное веб-приложение, которое работает внутри тега iframe, уведомления будут отображаться в маленьком окошке, что некрасиво и неудобно для пользователей. Ещё одним недостатком такого подхода является загрузка одного и того же кода дважды. Идеальным вариантом было бы как-то передавать данные из iframe в родительскую html-страницу.

В процессе изучения я нашёл решение: подписка на событие в родительском html, и генерирование события в iframe, современные браузеры умеют таким образом работать с событиями (*Таблица 7*). К сожалению, такой вариант не сработал из-за того, что viewer и фронтенд располагаются на разных доменах, а проброс события блокировался браузерной политикой одного источника (CORS).

Таблица 7

|  |
| --- |
| try {  // some unsafe code in iframe  } catch (e) {  window.parent.document.dispatchEvent(  new CustomEvent('notification/error', { detail: e })  )  }  window.document.addEventListener(  'notification/error',  e => { /\* handling in parent \*/},  false  ) |

Я продолжил искать пути решения. Оказывается, не так давно, разработчиками JavaScript был создан метод безопасного общения через кроссдоменные запросы – Window.postMessage(), который идеально подходит для моей задачи. Единственное ограничение – разрешена отправка только экземпляров класса String, а мне необходимо было отправлять экземпляры Error, но это ограничение легко обходится простой сериализацией в JSON. Чтобы сделать общение с фронтендом доступным из любого места viewer-а, я добавил метод отправки к прототипу глобального объекта Vue. Метод отправки уведомлений и его применение (*Таблица 8*).

Таблица 8

|  |
| --- |
| // function for sending data from iframe to parent  Vue.prototype.$notifyParent = (msg = '', type = 'notification/error') => {  window?.parent?.postMessage(JSON.stringify({ msg, type }), process.env.VUE\_APP\_FRONTEND\_URL)  }  // example of usage  const mtlxMaterial = await this.catchingLoadMaterial()  if (!Converter.isMtlx(mtlxMaterial)) {  this.$notifyParent(new Error(`Material '${this.materialPath}' not found on server`))  new FallbackMaterial(core, doc)  } else if (Converter.getMaterialVersion(mtlxMaterial) < 1.38) {  this.$notifyParent(new Error('3D mode supports only materials version 1.38 or above'))  new FallbackMaterial(core, doc)  } else await core.readFromXmlString(doc, mtlxMaterial)  // handling in parent  window.addEventListener(  'message',  e => {  if (event.origin === process.env.VUE\_APP\_URL\_VIEWER) {  const data = JSON.parse(event.data);  this.$store.dispatch(data.type, data.msg)  }  },  false  ) |

## 3.7 Добавить анимацию загрузки

На слабых компьютерах процесс смены меша/окружения и первоначальная загрузка занимают длительное время. Нужно добавить какую-нибудь анимацию загрузки, чтобы пользователю не показалось, что viewer просто завис.

Изначально я решил использовать для загрузки эффект полигональной сетки, реализованный ранее, будто скелет, обрастающий мясом (*Рисунок 11*). Для этого изменения необходимо было поменять конвейер рендеринга, чтобы отображать меш в процессе загрузки по частям, а не сразу; создать функцию рендеринга ограниченного количества кадров, чтобы не перегружать GPU во время загрузки (*Таблица 9*). Внешне, итоговый вариант ничем не отличается от соответствующего эффекта, тем не менее, команде 3D-artist результат не понравился: они выступили с критикой полигональной сетки существующих мешей.

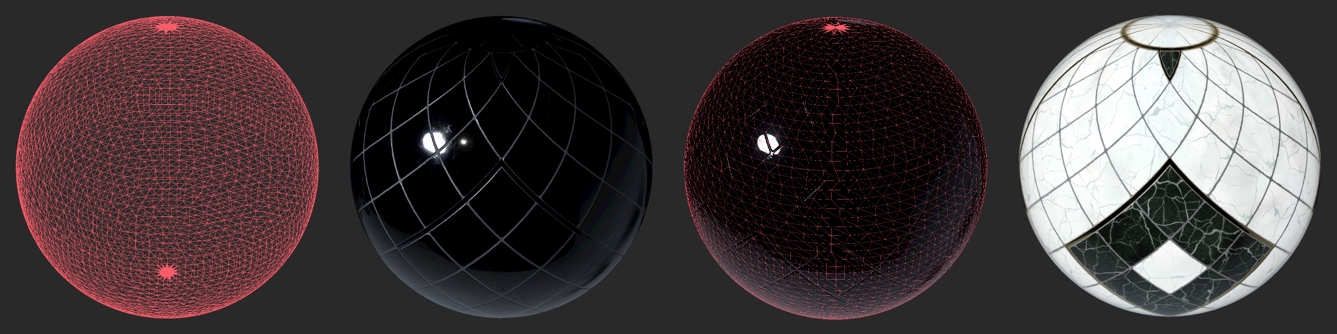


Рисунок 11

Таблица 9

|  |
| --- |
| methods: {  requestRender() {  if (!this.renderRequested) {  this.renderRequested = true  requestAnimationFrame(this.render)  }  },  render() {  this.renderRequested = false  this.composer.render()  this.model?.traverse( child => {  if (child.isMesh) {  const uniforms = child.material.uniforms  if (uniforms) // ... ligth and camera/mesh positions recompute  }  })  this.controls.update()  },  renderLimitedFrames(frames = 60, afterRender = () => {}) {  if (frames > 0) requestAnimationFrame(() => {  this.render()  this.renderLimitedFrames(frames - 1, afterRender)  })  else afterRender()  }  } |

Мы обсудили другие варианты и придумали тот, который визуально всем нравится, и даже не будет нагружать систему рендерингом сетки по время загрузки. Для первоначальной загрузки будет использоваться отдельный экран с прогресс-баром, как это сделано в играх, а для более короткой загрузки (смена меша/окружения), простое размытие экрана. Каких-то необычных сложностей в процессе создания не было, пример и реализация (*Рисунок 12*, *Таблица 10*).

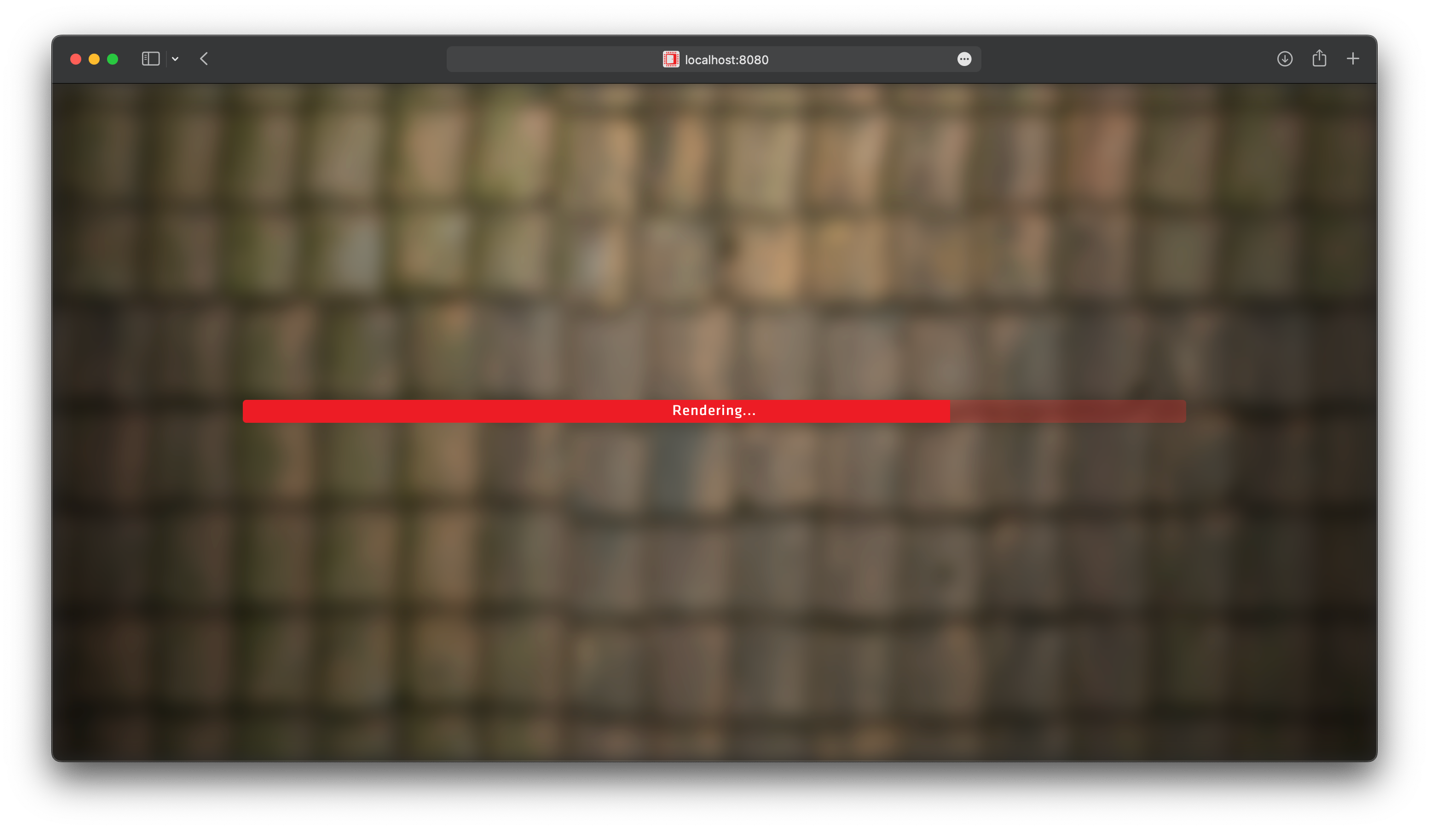


Рисунок 12

Таблица 10

|  |
| --- |
| <template>  <v-img  class="pulse"  :width="$vuetify.breakpoint.width"  :height="$vuetify.breakpoint.height"  :lazy-src="src"  eager  alt="baseColor.png from material package"  >  <template v-slot:placeholder>  <!-- don not use 'fill-height' class here cause problems on matlib.stvcis -->  <v-row  class="align-center justify-center"  style="height: 100%"  >  <div style="min-width: 70%">  <v-progress-linear  height="25"  rounded  :value="isLoaded ? 100 : Math.ceil(100 / steps.length) \* stepNumber"  >  <template v-slot:default>  <h3 class="ma-5 text-center">  {{ steps[stepNumber] }}  </h3>  </template>  </v-progress-linear>  </div>  </v-row>  </template>  </v-img>  </template>  <script>  export default {  name: "LoadingScreen",  props: {  steps: {  type: Array,  default: new Array()  },  src: {  type: String  }  },  data() {  return {  stepNumber: 1  }  },  methods: {  increment() { this.stepNumber++ },  reset() { this.stepNumber = 1 }  },  computed: {  isLoaded() { return this.stepNumber === this.steps.length }  }  }  </script>  <style lang="scss">  .pulse.v-image .v-image\_\_image {  filter: blur(10px) !important;  &::after {  content: '';  position: absolute;  height: 140%;  top: -40%; left: 0; bottom: 0; right: 0;  animation: pulse 1s infinite alternate;  background: radial-gradient(farthest-side at 50% -50%, transparent 50%, black 150%) no-repeat;  }  }  @keyframes pulse {  to {  top: 0;  }  }  </style> |

## 3.8 Реализовать управление направленным освещением

Добавить меню, позволяющие менять параметры направленного света (цвет, яркость, направление).

В отличие от глобального освещения, управлять которым мы не может по техническим причинам (функции расчёта инкапсулированы в другом модуле), мы можем менять направленное освещение, которое позволит пользователю смотреть, как выглядит тот или иной материал под определённым светом.

Для этого мною было создано отдельное плавающее окно (по аналогии с окном параметров материала), где располагаются элементы управления светом, создан новый источник света, добавлены контроллеры, написан метод обновления сцены по изменению света. Вся функциональность реализована средствами библиотеки Three.js, (пример *Рисунок 13*): так как в нашем случае рендерится только меш, а серый фон не является частью сцены (по желанию художников), то и изменения света видны исключительно на меше; металлический шар располагается под ярким фиолетовым светом.

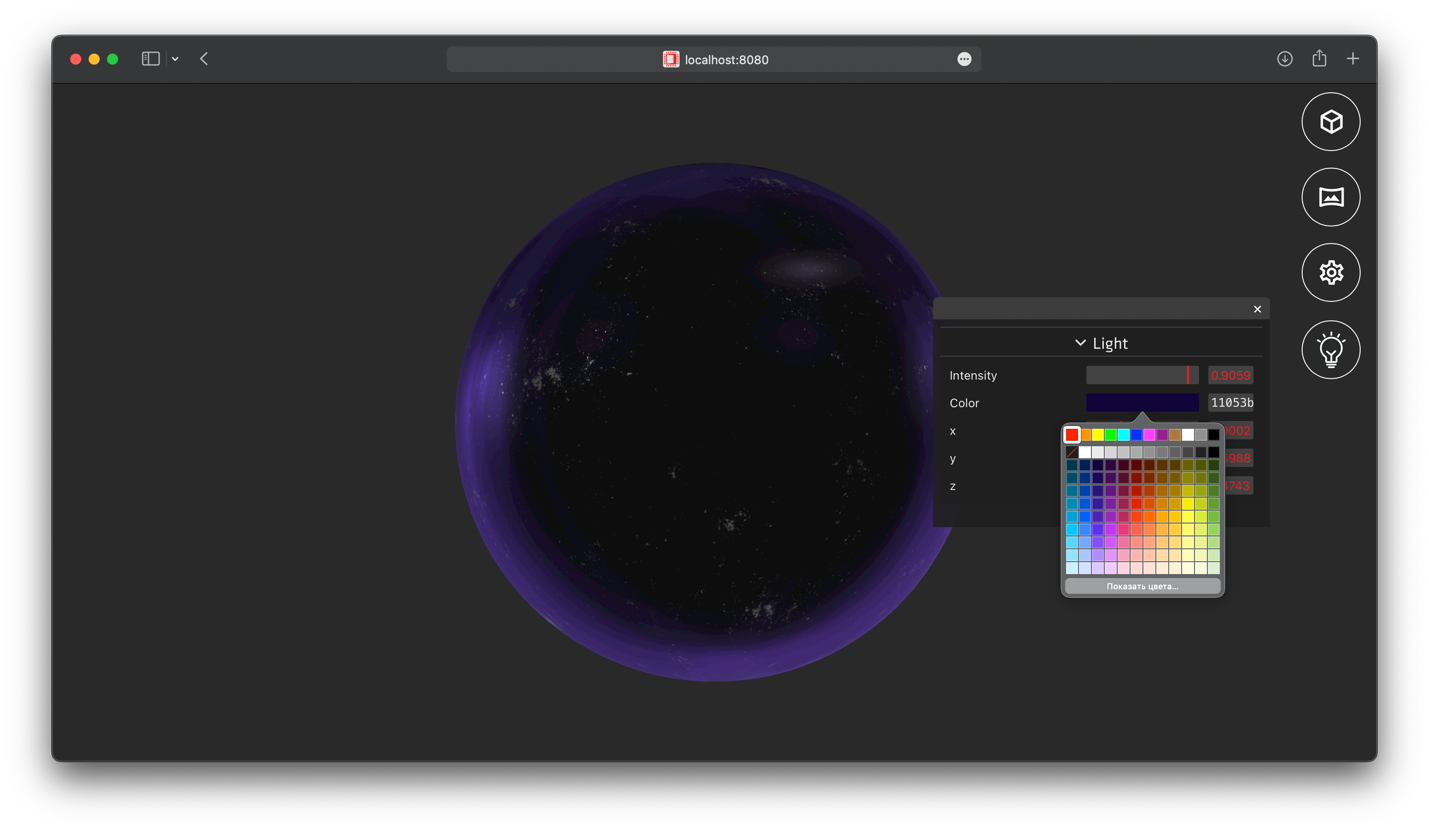


Рисунок 13

## 3.9 Мобильная версия сайта фронтенда matlib.gpuopen

На основе статистики использования сайта, собранной в период с декабря (релиз первой версии) по нынешний день, более трети пользователей заходили с смартфонов. Никто не ожидал, что на сайт, специфика которого предполагает компьютерное использование, будут заходить с мобильных устройств, а от того, некоторые специфичные ошибки были проигнорированы, а удобству использования на маленьких экранах внимания не уделяли. Необходимо просмотреть все страницы, агрегировать проблемы и исправить их.

Я начал просматривать страницы от корня сайта и за пару дней собрал отчёт из 13 пунктов. Обсудив с другими членами команды то, как должны выглядеть проблемные страницы на мобильных устройствах, я приступил к изменениям. Фреймворк Vuetify, используемый для построения интерфейсов имеет встроенную глобальную переменную $vuetify.breakpoints.mobile (и несколько переменных для конкретных размеров экранов), которая позволяет определить смартфон ли это, и динамически менять разметку. Пример кода определения разметки (*Таблица 11*), несколько примеров обновлённого вида в сравнении: новое меню категорий (*Рисунок 14*), карточка материала (*Рисунок 15*), меню управления аккаунтом (*Рисунок 16*).

Таблица 11

|  |
| --- |
| <template>  <!--...-->  <v-spacer v-if="$vuetify.breakpoint.mdAndDown && !totalRight"/>  <expandable-filter-menu  :expanded-class="expandedClass"  :is-expanded="$vuetify.breakpoint.lgAndUp"  v-if="!totalRight"  >  <slot v-if="!plotRight"/>  <slot name="actions-prepend"/>  <slot name="actions-left"/>  <v-spacer/>  <slot v-if="plotRight"/>  <slot name="actions"/>  </expandable-filter-menu>  <!--...-->  </template> |

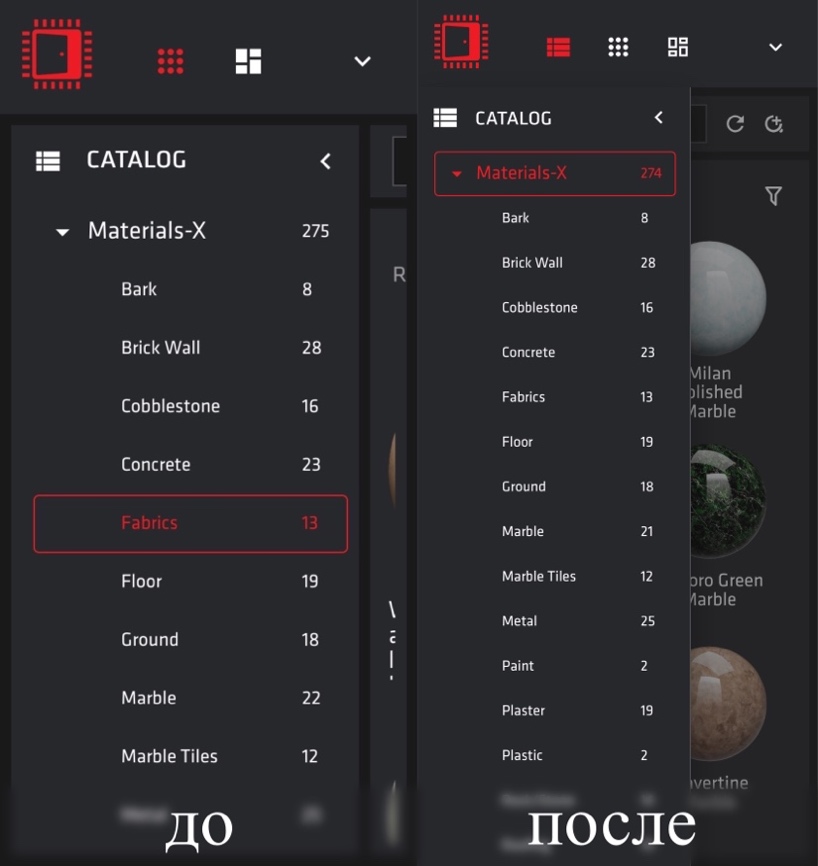


Рисунок 14



Рисунок 15

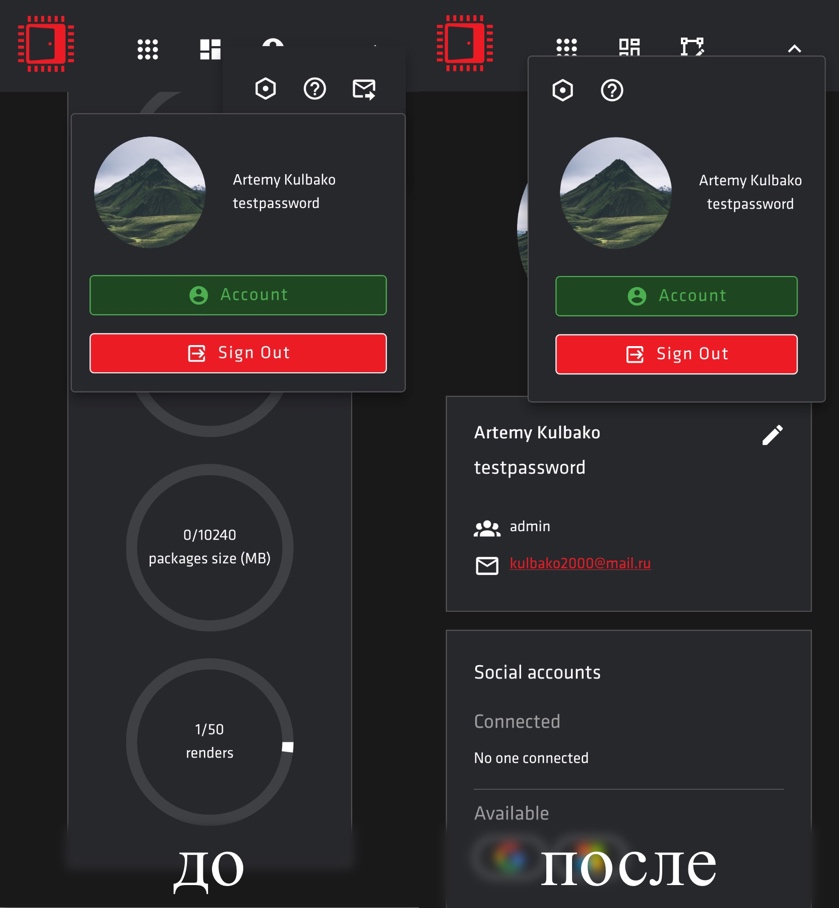


Рисунок 16

## 3.10 Исследовать поддержку WebGPU вместо WebGL

Необходимо изучить возможность использования новыго API взаимодействия браузера с GPU, что потенциально может улучшить производительность viewer-а.

В первую очередь я обратился к официальному сайту API [[14](#webgpu)] и изучил общую информацию о технологии. Далее я решил попробовать создать небольшой демо-проект на WebGPU. В процессе создания демо я столкнулся с необходимостью включать экспериментальный режим WebGPU в браузерах Firefox и Chrome (в Safari он и вовсе отсутствует), что уже показало несостоятельность использования технологии в проекте, так как заставлять пользователей менять настройки браузера идея гиблая. К тому же, библиотека Three.js, используемая в viewer-е расположена на более высоком уровне абстракции и скрывает все детали работы с API WebGL внутри класса WebGLRenderer. Попытка заменить WebGL на WebGPU сейчас, означала бы необходимость самому писать свой собственный класс-рендерер или целую графическую библиотеку, поэтому сейчас остаётся только ждать полноценного релиза WebGPU и его поддержки в Three.js. Я оформил всё эту информацию в виде подробного сообщения в Slack и отправил руководителю команды. Пример простейшего рендера с использованием WebGPU (*Рисунок 17*).

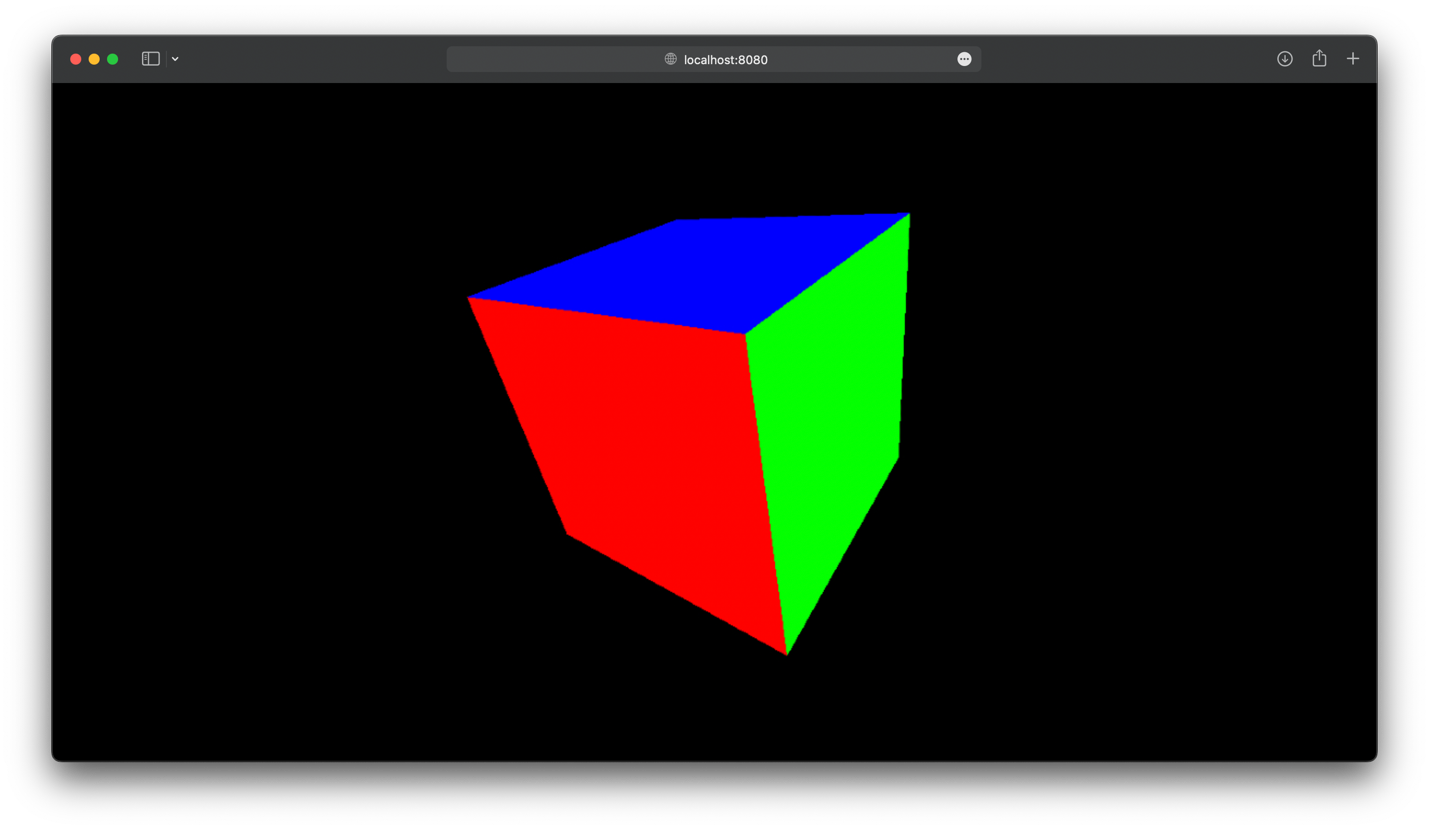


Рисунок 17

## 3.11 Добавить поддержку nodedef-нодов если есть поддержка ядром

Структурно, абстракция «материал», с которой работает viewer, представляет из себя xml-дерево, узлы которого описывают физические параметры материала (гладкость, прозрачность, металлизированность и т.д.), пример материала (*Таблица 12*). Менять эти параметры можно динамически в меню параметров viewer-а с помощью написанных самостоятельно контроллеров. Помимо основных параметров, определённых спецификацией MaterialX, художники могут определять свои собственные узлы, в файле они представлены xml-тегом nodegraph, который должен иметь определённую структуру. В новой версии спецификации (1.38.3) узел nodegpaph заменили на nodedef, а структуру узла поменяли. За обработку узлов отвечает отдельное ядро viewer-а, разрабатываемое open-source командой фирмы Autodesk. Необходимо проверить, реализована ли поддержка nodedef в ядре, и, если ответ положительный, написать конструктор контроллеров для управления этими узлам.

Таблица 12

|  |
| --- |
| <?xml version="1.0"?>  <materialx version="1.38" xmlns:xi="http://www.w3.org/2001/XInclude" fileprefix="public/materials/demo/textures/">  <xi:include href="stdlib/stdlib\_defs.mtlx" />  <!-- ... -->  <nodegraph name="NDMaterialX\_Graph" node="MaterialX\_Graph">  <output name="base\_color\_output" type="color3" />  <output name="metalness\_output" type="float" />  <output name="specular\_roughness\_output" type="float" />  <output name="coat\_normal\_output" type="vector3" />  <output name="normal\_output" type="vector3" />  <output name="tangent\_output" type="vector3" />  <input name="UVScale\_2" type="vector2" value=" 1.000000, 1.000000" uisoftmin="0.0,0.0" uisoftmax="1.0,1.0" uimin="0.0,0.0" uimax="1.0,1.0" uniform="true" />  <input name="RoughnessHigh" type="float" value="1.0" uisoftmin="0.0" uisoftmax="1.0" uimin="0.0" uimax="1.0" uniform="true" />  <input name="RoughnessLow" type="float" value="0.07000000029802322" uisoftmin="0.0" uisoftmax="1.0" uimin="0.0" uimax="1.0" uniform="true" />  </nodedef>  <standard\_surface name="MaterialX\_Graph2" type="surfaceshader">  <input name="base" type="float" value="0.800000011920929" />  <!-- ... -->  </standard\_surface>  <!-- ... -->  </materialx> |

Для начала необходимо обратиться к репозиторию, в котором ведётся разработка ядра [[15](#matx_autodesk)]. К сожалению, разработчики не ведут какой-либо истории изменений, а сообщения к коммитам не информативны. Проверить поддержку путём изучения исходного кода для меня слабо представляется возможным, так как код ядра написан на C++ с использованием WebAssembly, технологии, с которыми я лишь очень поверхностно знаком. Да и в случае, если поддержка всё-таки есть, собирать проект всё равно придётся, поэтому сразу к этому и приступил. Для сборки модуля необходимо установить ряд программ: CMake, Emscripten, Python 3 и любой современный компилятор C++, а после запустить cmake-скрипт в корне директории с исходниками для JavaScript. Строгое следование документации, спасло меня от всех проблем, кроме одной, специфичной для моего оборудования: так как я использую ноутбук с процессором на архитектуре ARM, а не x86\_64, то Python-скрипт, внутри CMake скрипта выдавал ошибку при компиляции одного из компонентов. Проблему решил путём принудительного запуска всех программ (в том числе и Python) в режиме трансляции x86 в текущем терминале командой: arch -x86\_64 zsh.

В результате сборки я получил пять файлов, код с расширением wasm и обёртки с расширение js, заменил старые файлы в проекте на новые. После, взял материал, где присутствуют пользовательские узлы и вывел в лог содержание объекта, который занимается маршалингом этих узлом (*Таблица 13*). К сожалению, поддержки нет, я уведомил об этом руководителя и было принято решение отложить внедрение управлением nodedef-узлов.

Таблица 13

|  |
| --- |
| {  // ...  "node\_image\_vector3\_8\_file": {  "value": {  "metadata": {  "version": 4.5,  "type": "Texture",  "generator": "Texture.toJSON"  },  "uuid": "4BB386CB-E9A3-41DF-A80B-DB2FCAC3783B",  "name": "",  "mapping": 300,  "repeat": [  1,  1  ],  "offset": [  0,  0  ],  "center": [  0,  0  ],  "rotation": 0,  "wrap": [  1000,  1000  ],  "format": 1023,  "type": 1009,  "encoding": 3000,  "minFilter": 1008,  "magFilter": 1006,  "anisotropy": 1,  "flipY": true,  "premultiplyAlpha": false,  "unpackAlignment": 4  }  },  // ...  } |

## 3.12 Оформление отчётности

Последний этап заключался в написании данного отчёта: пользуясь утилитой git я возвращался к произведённым мною изменениям в проекте, делал скриншоты и вырезал код для примеров. Так как у меня уже имелся опыт оформления отчёта с учебной практики, то моя скорость значительно возросла, сложности в процессе не возникли.

# 4. ЗАКЛЮЧЕНИЕ

Цель практики – продемонстрировать наличие навыков, умений и компетенций для работы по специальности была достигнута: я выполнил все поставленные передо мной задачи, руководитель и другие члены команды остались довольны результатом.

Я считаю, что получил большой опыт в рамках практики, так как работать приходилось с слабо знакомой мне библиотекой Three.js и в сфере компьютерной графики, что я нахожу очень интересным. При этом, пути выполнения заданий не регламентированы, что давало простор для творчества. Помимо этого, улучшил свои навыки вёрстки и программирования для веба, дисциплину через посещение митингов и регулярное чтение рабочих чатов и почты. Произведённые мною изменения войдут в следующую версию сайта и будут доступны на <https://matlib.gpuopen.com>.

# 5. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Программирование компьютерной графики. Современный OpenGL./ Боресоков А. В. – ДМК Пресс, 2019 – 372 с.
2. Learning Three.js - the JavaScript 3D Library for WebGL / Дирскен Й. – Packt Publishing, 2015 – 422 с.
3. WebGL. Программирование трехмерной графики / Мацуда К., Ли Р. – ДМК Пресс, 2015 – 494 с.
4. Выразительный JavaScript / Хавербаке М. – Издательство Питер, 2019 – 480 с.
5. Большая книга CSS3 / Макфарланд Д. – Издательство Питер, 2016 – 608 с.
6. Современный учебник JavaScript / Кантор И. [Электронный ресурс] – URL: <https://learn.javascript.ru/>
7. Документация библиотеки Three.js [Электронный ресурс] – URL: <https://threejs.org/>
8. Документация JavaScript, HTML, CSS от Mozilla [Электронный ресурс] – URL: <https://developer.mozilla.org/ru/>
9. Статья, инструкции и новости IT-сферы [Электронный ресурс] – URL: <https://habr.com/>
10. Документация фреймворка Vuetify [Электронный ресурс] – URL: <https://vuetifyjs.com/>
11. Репозиторий для JavaScript [Электронный ресурс] – URL: <https://www.npmjs.com/>
12. Глобальный форум разработчиков [Электронный ресурс] – URL: <https://stackoverflow.com/>
13. Документация фреймворка Vue [Электронный ресурс] – URL: <https://vuejs.org/>
14. Черновик стандарта WebGPU [Электронный ресурс] – URL: <https://www.w3.org/TR/webgpu/>
15. Исходный код и документация ядра viewer-а от Autodesk [Электронный ресурс] – URL: <https://github.com/autodesk-forks/MaterialX>