**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](#_Toc98694319)

[1. ВВЕДЕНИЕ 3](#_Toc98694320)

[2. ХАРАКТЕРИСТИКА ОРГАНИЗАЦИИ 4](#_Toc98694321)

[3. ОПИСАНИЕ ВЫПОЛНЕНИЯ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ 5](#_Toc98694322)

[3.1 Исправить существующие ошибки и недочёты 5](#_Toc98694323)

[3.3 Сделать окно параметров плавающим 6](#_Toc98694324)

[3.4 Заменить проблемные меши на меши с правильной UV 6](#_Toc98694325)

[3.5 Добавить поддержку различных эффектов постобработки 7](#_Toc98694326)

[3.6 Имплементировать систему уведомлений об ошибках с основного фрондента matlib.gpuopen 7](#_Toc98694327)

[3.7 Добавить анимацию загрузки 8](#_Toc98694328)

[3.8 Реализовать управление направленным освещением 8](#_Toc98694329)

[3.9 Мобильная версия сайта фронтенда matlib.gpuopen 8](#_Toc98694330)

[3.10 Исследовать поддержку WebGPU вместо WebGL 8](#_Toc98694331)

[3.11 Добавить поддержку nodedef-нодов если есть поддержка ядром 8](#_Toc98694332)

[3.12 Оформление отчётности 8](#_Toc98694333)

[4. ЗАКЛЮЧЕНИЕ 8](#_Toc98694334)

# 1. ВВЕДЕНИЕ

Производственная практика – заключительный этап образовательного процесса, направленный на проверку и закрепление

Производственная практика – 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 об ошибках 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

ПОПРАВИТЬ ОТСТУПЫ МЕЖДУ АБЗАЦАМИ ВО ВСЁМ ДОКУМЕНТЕ

КАРТИНИКИ И ЛИСТИНГИ

Первая.

Вторая.

# 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 скроллбара стилю сайта. Я просто забыл про него. Проблема решилась простым подключением файла стилей в проект РИСУНОК1; ЛИСТИНГ 1.

Вторая – артефакты на краях мешей – яркие пиксели, образующие эффект «лесенки», появившиеся из-за четырёхкратно возросшего разрешения рендера РИСУНОК 2. 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 3; 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.

**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 ЛИСТИНГ 2 https://stackoverflow.com/questions/13898684/sub-pixel-antialiasing-canvas-pixel-shift-algorithms-in-javascript). 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 4) (разница видна в основном в динамике).

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 5). Команда 3d-artist предложила разрешить свободно перемещать его по экрану. Задача нетривиальная, так как теги html, Vue или компоненты Vuetify подобной функциональности не представляют. В первую очередь я принялся искать готовое решение в репозитории пакетов для JavaScript – npm и там мною был найден подходящий компонент (<https://www.npmjs.com/package/@hscmap/vue-window> ). 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 (ЛИСТИНГ 3). В конце необходимо было внести небольшие правки по дизайну, предложенные другими членами команды на общем обсуждении. Новый вариант на РИСУНКЕ 6.

## 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 в 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.

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 на ЛИСТИНГЕ 4, а пример меша с неправильной и правильной развёртками на РИСУНКЕ 7.

## 3.5 Добавить поддержку различных эффектов постобработки

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

Сначала я изучил список эффектов, которые поддерживает Three.js (<https://threejs.org/docs/#examples/en/postprocessing/EffectComposer>). Основная проблема заключается в том, что ен все доступные эффекты будут работать ввиду того, что часть из ни должна взаимодействовать с шейдерами, отвечающими за просчёт света, а в нашем случае, этим занимается инкапсулированный код, подключаемые в виде отдельного WebAssembly модуля. Другая часть эффектов просто бессмысленна, к примеру, глубина резкости, так как за мешем мы не отображаем фон. Мне нужно было выбирать эффекты, которые не относятся к двум перечисленным группам.

В итоге, я реализовал несколько эффектов: «глитч», отображение полигональной сетки, имитация зернистости плёнки, SSAO – улучшенные тени. Два последние самые красивые, примеры на РИСУНКЕ 8 и РИСУНКЕ 9. Пример реализации одного из эффектов – отображение сетки в ЛИСТИНГЕ 5.

## 3.6 Имплементировать систему уведомлений об ошибках 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).

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 является загрузка одного и того же кода дважды. Идеальным вариантом было 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 (ЛИСТИНГ 6). К сожалению, такой вариант не сработал ввиду того, что viewer и фронтенд располагаются на разных доменах, а проброс события блокировался политикой одного источника (CORS). 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 на ЛИСТИНГЕ 7.

## 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 загрузку по частям, а не сразу; создать функцию рендеринга ограниченного количества кадров, чтобы не перегружать GPU во время загрузки (ЛИСТИНГ 8). 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.

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 на РИСУНКЕ 12 и ЛИСТИНГЕ 9.

## 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 инкапсулировано 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 работы на РИСУНКЕ 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.

## 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 почти треть 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 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 представлен в ЛИСТИНГЕ 10, несколько примеров обновлённого вида в сравнении: новое меню категорий (РИСУНОК 14), карточка материала (РИСУНОК 15), меню управления аккаунтом (РИСУНОК 16).

## 3.10 Исследовать поддержку WebGPU вместо WebGL

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

В первую очередь я обратился к официальному сайту API – <https://www.w3.org/TR/webgpu/> и изучил общую информацию о технологии. Далее я решил попробовать создать небольшой демо-проект на WebGPU по статье с сайта habr.com (<https://habr.com/ru/company/ruvds/blog/485644/>). В процессе создания демо я столкнулся с необходимостью включать экспериментальный режим WebGPU в браузере (а в Safari он и вовсе отсутствует), что уже показало несостоятельность использования технологии в проекте, так как заставлять пользователей менять настройки или браузер идея гиблая. К тому, 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 свою собственную графическую библиотеку, поэтому сейчас остаётся только ждать полноценного релиза WebGPU и его поддержки в Three.js. Я оформил всё эту информацию в виде сообщения в Slack и передал руководителю команды. Пример простейшего рендера с использованием WebGPU на РИСУНКЕ 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 в ЛИСТИНГЕ 11. Менять эти параметры можно динамически в меню параметров viewer 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 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.

Для начала необходимо обратиться к репозиторию, в котором ведётся разработка ядра (<https://github.com/autodesk-forks/MaterialX>). К сожалению, разработчики не ведут какой-либо истории изменений, а сообщения к коммитам информативно пусты. Проверить поддержку путём изучения исходного кода для меня слабо представляется возможным ввиду того, что код ядра написан на C++ с использованием WebAssembly, технологии, с которыми я лишь очень поверхностно знаком. Да и в случае, если поддержка всё 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 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 необходимо взять материал, где присутствуют пользовательские узлы и вывести в лог содержание объекта, который занимается маршалингом этих узлом (пример в ЛИСТИНГЕ 12). К сожалению, поддержки нет, я уведомил об этом руководителя и было принято решение отложить внедрение управлением nodedef-узлов.

## 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.