Лабораторная работа 7

Практикум по созданию интерактивной цветной 3D графической модели для web-browser на основе цветной цифровой 2D фотографии с применением нейронных сетей, графического редактора Blender, языка для структурирования и отображения содержимого «всемирной паутины» HTML-5, библиотеки WEBGL для синтеза 2D/3D графики в web-browsers.

Цель. Овладение практическими навыками по применению нейронных сетей, графического редактора Blender, WEBGL и HTML-5 по созданию web-страниц с интерактивными цветными 3D объектами на основе цветных цифровых 2D фотографий.

Информационные ссылки

1. Blender // https://www.blender.org/
2. HTML 5 // https://html.spec.whatwg.org/multipage/
3. WEBGl // https://www.khronos.org/webgl/
4. Нейронная сеть // https://shunsukesaito.github.io/PIFuHD/
5. Google Colab // https://colab.research.google.com/
6. Видео справочное // https://www.youtube.com/watch?v=ZzVNscwMzBE
7. Visual Studio Code // https://code.visualstudio.com/
8. Docker // https://www.docker.com/products/docker-desktop/

Задание

1. Создание высококачественной фотографии самого себя

Необходимо создать цветную высококачественную с высокой степенью детализации цифровую фотографию **в полный рост самого себя** в формате **png** или **jpg** размером с 1024x1024 или 512x512 при помощи цифрового фотоаппарата или смартфона. Примеры **требуемых** поз при фотографировании приведены в [4] в пункте «Results (Internet Photos)», а также в приложении в файлах PNG к этой лабе. Можно сделать фото в T-Pose для наилучшего результата. Убедитесь, что изображение хорошо освещено. Чрезвычайно темное или яркое изображение и сильные тени часто создают артефакты. Фон должен быть простым или однотонным. Фон вообще можно удалить при помощи GIMP или другой любой подобной графической программы.

2. Преобразование фотографии в 3d модель с помощью нейронной сети

Используя нейронную сеть [4], Google Colab и Jupyter Notebook из Google Colab [5], необходимо создать 3D модель в формате **.obj** по вашей фотографии из пункта 1.

Зайдите на сайт нейронной сети [4] и в самом низу перейдите по ссылке Google Collab. После этого у вас должен будет открыться блокнот «PIFuHD Demo», как показано на рис.1:

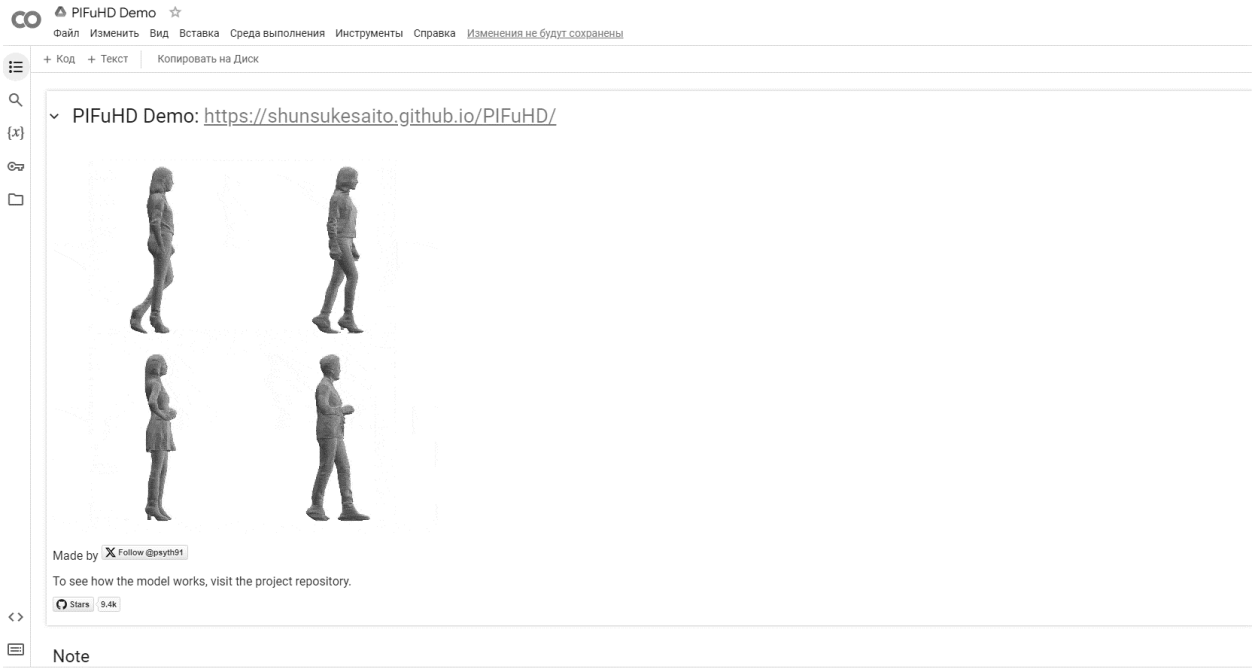

Рис. 1. PIFuHD Demo

При запуске нейронной сети необходимо использовать для ускорения ее работы аппаратный ускоритель **GPU.** Вверху, среди пунктов: Файл, Изменить, Вид, Вставка, Среда выполнения, Инструменты и Справка, выберете следующее: **«Среда выполнения → Сменить среду выполнения → Аппаратный ускоритель → GPU»**.

Прокрутите страницу вниз и найдите пункт «**Clone PIFuHD repository**». Отсюда нужно будет начать запускать все ячейки. Запустите ячейку **«!git clone https://github.com/facebookresearch/pifuhd**». Следующим идет пункт «**Configure input data**». Здесь нужно будет запустить только ячейку под заголовком «**If you want to upload your own picture, run the next cell. …».** Запустите ее, после запуска можно будет нажать «**Выбрать файлы**», выберете свое фото.

В ячейке «**import os**» укажите в переменных «image_path = '/content/pifuhd/sample_images/**test.png»** вместо **test.png** имя вашего файла с фото п.1. Запустите эту и все остальные ячейки, идущие далее.

Если у вас возникает ошибка, в пункте «**Preprocess (for cropping image)»,** в ячейке, где идет импорт модулей (Torch, Numpy, Cv2 и т.д.), замените «np.int» на «np.int_», т.к. «np.int» на данный момент устарел. Подробнее здесь.

После завершения работы нейронной сети файл 3D модели фото в формате **«.obj»** будет находиться в папке **recon** (**иконка «Файлы»** (панель слева), затем надо пройти по структуре дерева файлов **«pifuhd → results → pifuhd_final»** для доступа к папке **recon**. Выбираем целевой файл, нажимаем на **3 точки справа**, выбираем пункт скачать и скачиваем на свой компьютер.

3. Редактирование полученной модели в Blender

Используя Blender, версия 3.4 и выше , [1], необходимо создать окончательно цветную 3D модель фото из п. 1. Блендер можно скачать на сайте[1] или из Steam.

После запуска Blender необходимо удалить со сцены все лишнее. Для этого нажмите «A», чтобы выбрать все, а затем клавишу «**Delete»**. Для загрузки файла «**.obj»** используется «**File →Import →Wavefont(.obj)».** Найдите вашу модель и импортируйте ее в Blender.

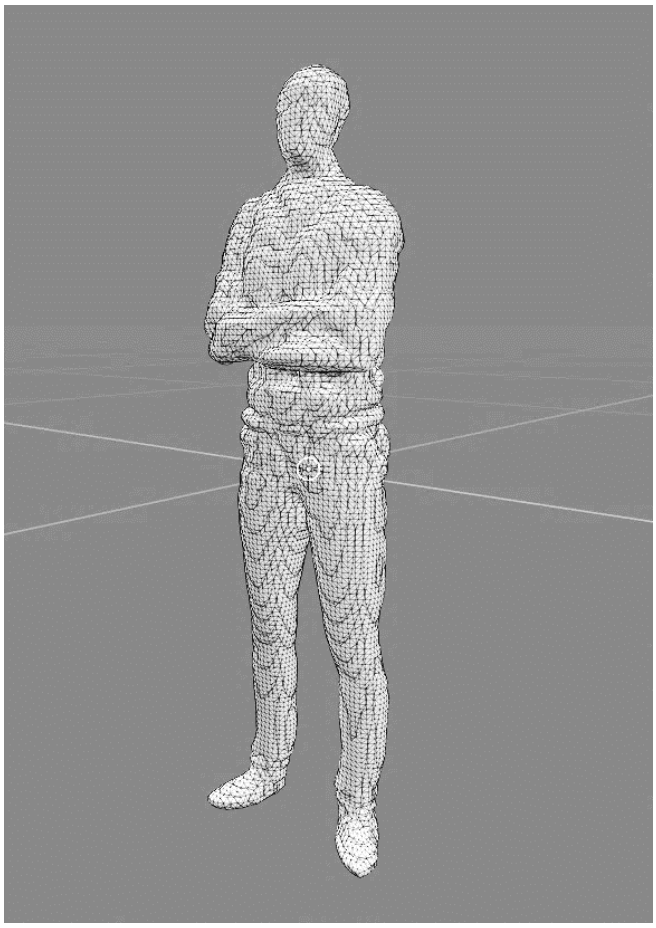

Рис. 2. Экспортированная модель в режиме Edit Mode с большим количеством полигонов

Выберете импортированную модель (будет подсвечиваться оранжевым) и войдите в режим редактирования **«Edit Mode»** (клавиша **«Tab»**). Нажмите цифру **«3»** на клавиатуре, для отображения поверхностей полигонов. Если вы видите слишком большое количество треугольников/прямоугольников, значит модель высокополигональная или просто имеет большое количество полигонов, что может затруднить создание развертки для текстурирования в будущем.

Количество полигонов можно уменьшить без ущерба исходному внешнему виду, посредством создания карты нормалей. Подробнее о том, как работают карты нормалей, можно прочесть здесь. Карты нормалей часто используются в гейм-дизайне, в условиях, когда количество полигонов ограничено. Мы воспользуемся одним из множества возможных способов для их создания. Нам понадобятся две версии модели: высокополигональная - исходная и низкополигональная - измененная.

Выйдите из режима редактирования **«Edit Mode»** и переименуйте (клавиша **«F3»** или двойной клик по старому названию на панели со слоями справа) исходный меш на ««HighPoly»». Скопируйте «HighPoly» с помощью клавиши **«Shift+D»**, после нажмите **«Esc»**, чтобы скопированная модель осталась в том же месте, что и исходная. Переименуйте скопированный объект на «LowPoly». Обе модели должны находиться в одном и том же месте, перекрывая друг друга. Скройте «HighPoly» и выберите «LowPoly», как показано на рис.3

Рис. 3. «LowPoly» и «HighPoly»

> Если в режиме редактирования **«Edit Mode»** вы видите, что модель имеет щели и разрывы, можно сделать **«Remesh»** с низкими значениями **«Voxel Size»** в режиме **«Sculpt Mode»**, для исправления. В таком случае, сделать это необходимо до создания копии исходной модели.

Выбрав «LowPoly», справа, в панели **«Properties»**, войдите во вкладку **«Modifiers»** (иконка синего гаечного ключа). Нажмите **«Add Modifier»** и найдите в поиске модификатор **«Decimate»**. Установите значение **«Ratio»** так, чтобы исходная форма не сильно изменилась, но количество полигонов заметно уменьшилось. Влияние значения **«Ratio»** зависит от исходного количества полигонов. Примените модификатор.

Создадим развертку «LowPoly» модели. Она понадобится для карты нормалей и текстурирования. Выберем «LowPoly» модель с уменьшенным количеством полигонов, перейдем в **«Edit Mode»**, а затем во вкладку **«UV Editing»** на панели сверху. Перед вами должно открыться следующее окно:

Рис. 4. UV Editing

Переведите курсор в первоначальное окно **«3D Viewport»** и нажать на клавишу **«Tab»** (переход в **«Edit Mode»**), затем на клавишу **«A»** (выделить все). 3D модель должна поменять цвет с черного на «рыжеватый». Из контекстного меню (после нажатия клавши **«U»**) выбрать **«Project from view»**, после чего маска изображения из окна **«3D viewport»** появиться в окне **«UV Editing»** слева. Далее в этом окна выбираем пункт **«Open»** и загружаем файл с исходным фото п.1.

Теперь UV-развертку нужно будет отредактировать. Но для начала, для удобства, сделаем отображения текстуры на 3d объекте. Выберем нашу «LowPoly», перейдем в Layout. По умолчанию, в нижней панели отображается **«Timeline».** Чтобы поменять ее на

любую другую среду (в нашем случае понадобится **«Shader Editor»**), кликнем на самую верхнюю правую иконку в окне **«Timeline»**. Здесь выбираем **«Shader Editor»**.

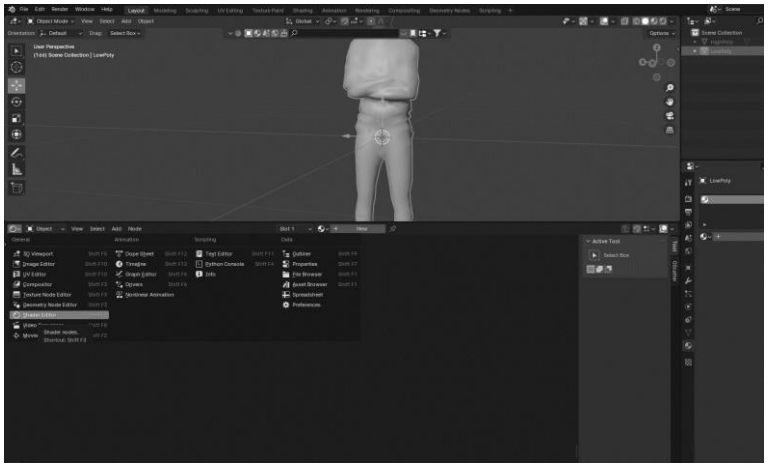

Рис. 5. Shader Editor

Теперь нажмем на кнопку **«New»** сверху в центре окна **«Shader Editor»**, появится две плитки (нода): **«Principled BSDF»** и **«Material Output»**. Нажмем **«Shift+A»** и в поиске введем **«Image Texture»**. В ноде **«Image Texture»**, кликните по иконке картинки слева (раскрывающийся список) и выберите файл с исходным фото, который вы уже загрузили ранее (Имя_файла.png).

После этого необходимо соединить точку Color в **«Image Texture»** с Base Color в **«Principled BSDF»**:

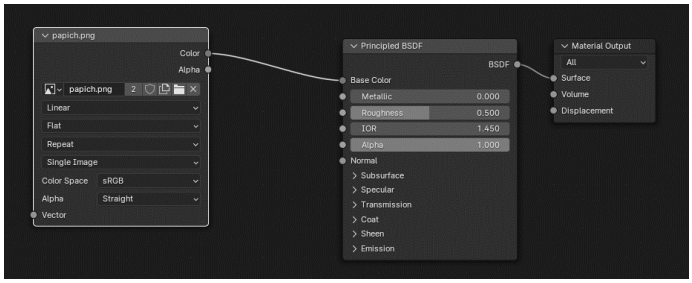

Рис. 6. Соединение нодов

Переходим в окно (перемещаем указатель мыши) **«3D Viewport»**, кликаем **«Z»** и из всплывающего меню выбираем **«Material Preview»** или **«Rendered»**. Если модель выглядит

темной из-за недостатка источника света, В окне Viewport сверху справа, нажмите на иконку раскрывающегося списка. Уберите галочки **«Scene Lights»** и **«Scene World»**:

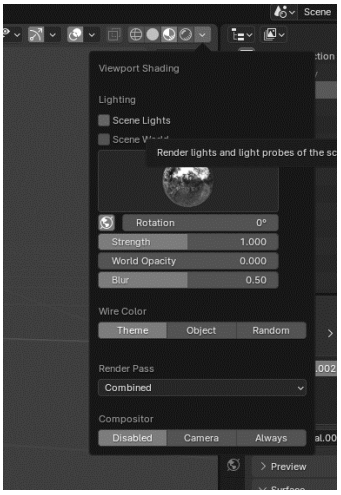

Рис. 7. Предпросмотр без учета окружения сцены

Вернемся в **«UV Editing»**, выбрав «LowPoly», нажмем **«Tab»** и выделим все. Нажмите клавишу **«1»** на клавиатуре, чтобы отобразить точки, вместо поверхностей (1 – точки, 2 – ребра, 3 – поверхности). Теперь, используя инструменты увеличения масштаба, вращения и перемещения добиваемся того, чтобы загруженная маска изображения была бы максимально точно подогнана по размерам к загруженному исходного фото. Нажатие (не зажатие, а просто нажать и отпустить) клавиш G , R , S, L выполняют операции:

G – grab/move – перемещение

R – rotate – вращение

S – scale – изменение размера

L – выбрать кусок развертки

Можно выходить и возвращаться в режим редактирования **(«Tab»)** для удобства просмотра результата:

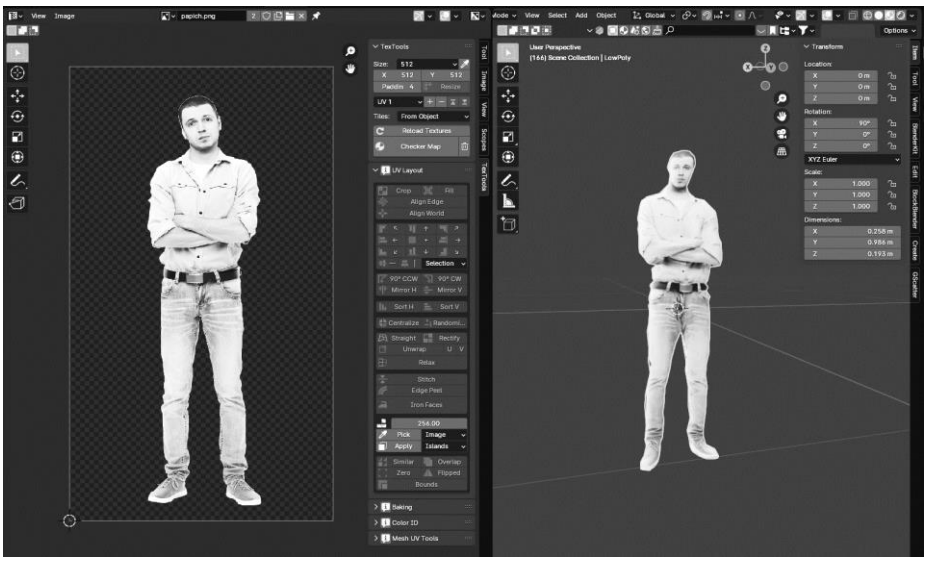

Рис. 8. Результат текстурироания

Добавим карту нормалей. Вернитесь в **«Layout»**, внизу у вас должна быть по-прежнему открыта панель **«Shader Editor»**. Здесь, нажмите **«Shift+A» → «Image Texture»**. Кликните **«New»** и создайте следующую текстуру с названием «NormalMap»:

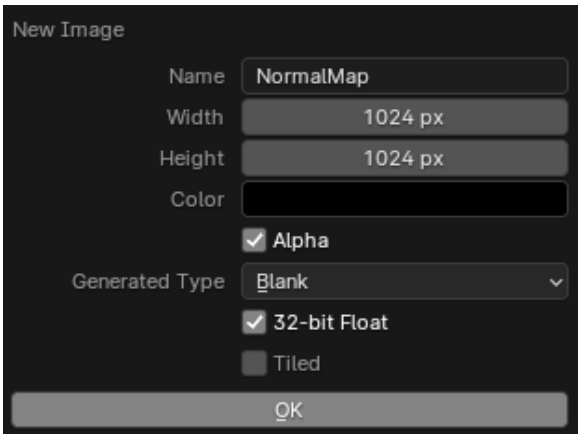

Рис. 9. Карта нормалей

После нажатия OK, в созданном ноде «NormalMap» выберите **«Color Space» → «Non-Color»**. Добавьте еще один нод, **«Shift+A» → «Normal Map»**. Соедините ноды как показано на рисунке:

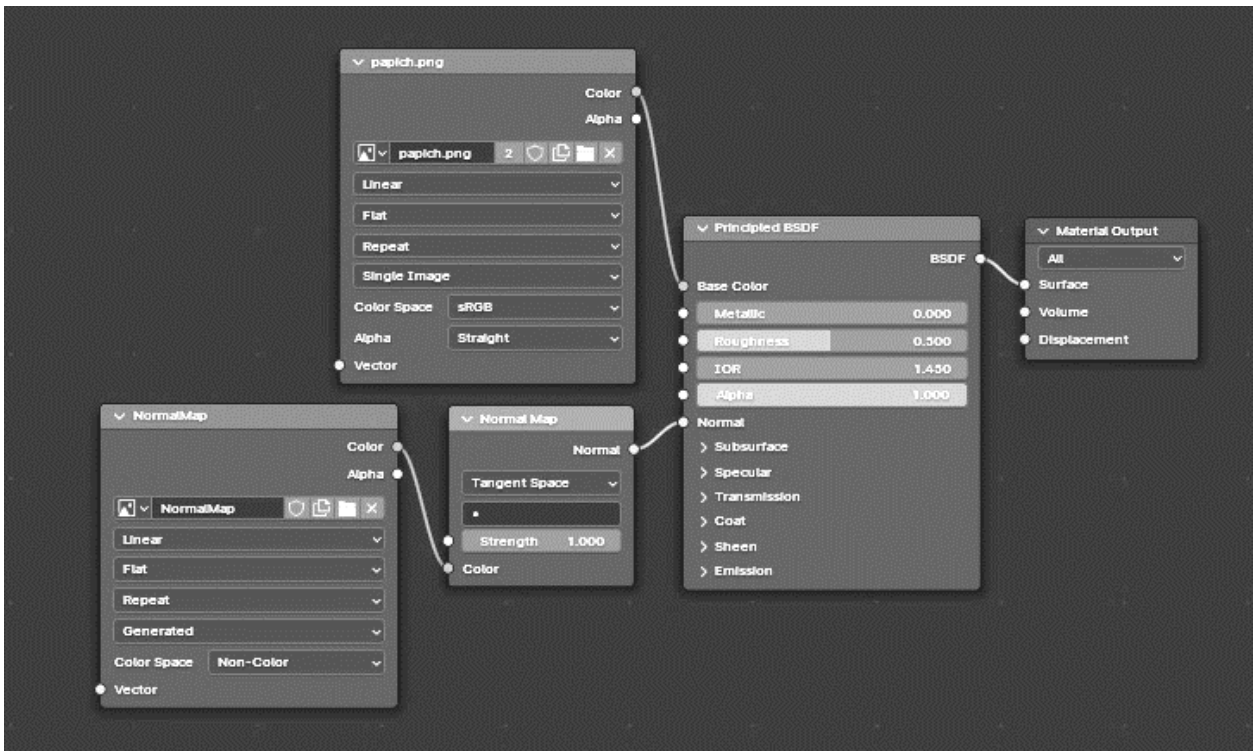

Рис. 10. Соединение нодов в Shader Editor

Теперь сделайте видимой модель ««HighPoly»», она должна перекрывать затекстурированную модель. ПКМ по ««HighPoly»» → **«Shade Smooth»**. Перейдите в **«Properties»**, во вкладку **«Render»** (иконка телевизора). Поменяйте **«Render Engine»** на **«Cycles»**.

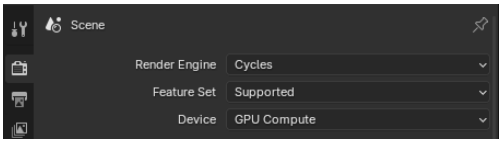

Рис. 11. Выбор движка для рендера

Прокрутите чуть ниже и найдите во вкладке **«Render»** пункт **«Bake»**, раскройте его. Установите следующие параметры:

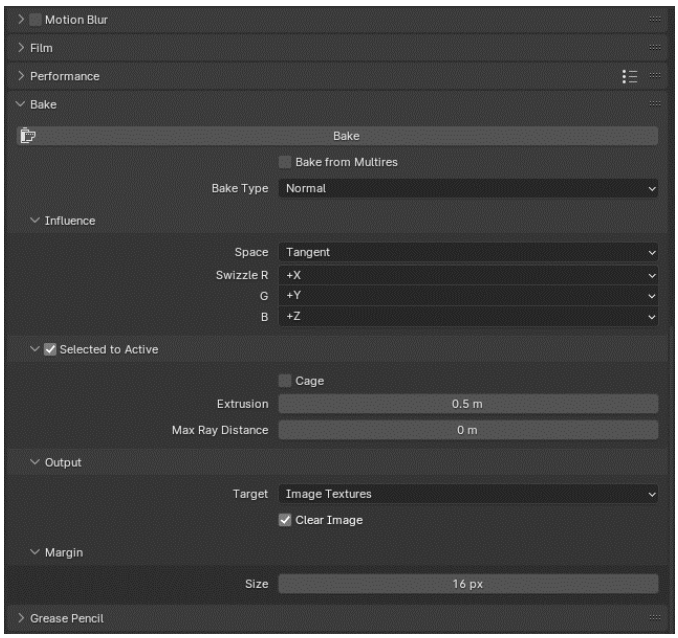

Рис. 12. Bake

Выберите «HighPoly», далее, зажав клавишу **«Ctrl»**, выберите ««LowPoly»». После, во вкладке **«Shader Editor»** ««LowPoly»» модели, кликните по ноду с названием ««NormalMap»», чтобы он был активным.

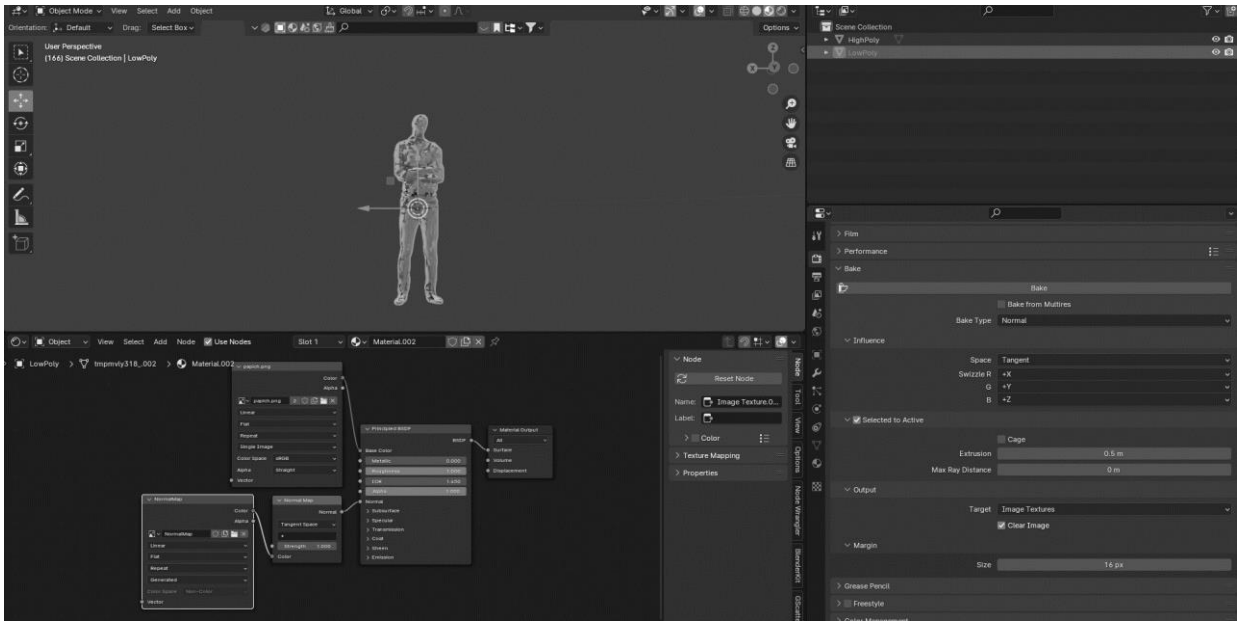

Рис. 13. Бейкинг карты нормалей

После нажмите кнопку **«Bake»** справа. Спустя некоторое время, появится карта нормалей. Скройте ««HighPoly»» модель и оцените результат, нажав **«Z»** → **«Rendered»**. Посмотреть, как выглядит карта нормалей можно во вкладке **«UV-Editing»**. Она должна быть фиолетового оттенка. Можно удалить ««HighPoly»», если все устраивает. Если вас не

устроил результат, попробуйте изменить параметры бейкинга или саму ««HighPoly»» модель. Подробнее об этом можно прочесть здесь.

Результат готов.

Полученный результат сохраняем в файле с расширением **«.blend»**, а также экспортируем с расширением «**.glb»**. В появившемся диалоговом окне экспорта убедитесь, что опция "Include Texture" (Включить текстуры) включена. Обычно по умолчанию она включена, но всегда лучше это проверить. Текстуры и карты нормалей в формате GLB (Binary glTF) включены непосредственно в файл модели.

4. Демонстрация результата

Разработаем 3D Viewer для формата **«.glb»** на базе HTML5 и WEBGl [2,3], обеспечивающего масштабирование и вращение.

Создайте новую папку на вашем компьютере для вашего проекта. Откройте Visual Studio Code [7]. Нажмите «File» → «Open Folder» и выберете созданную папку проекта. Теперь на панели слева вам доступно для просмотра и манипуляций содержимое папки. Создайте файлы «**script.js»** и **«index.html»** с помощью иконки файла или ПКМ по области → «New File».

Сюда же, в эту папку, переместите файл вашей модели с расширением **«.glb».** Переименуйте файл с расширением **«.glb»**. на **«model.glb»**.

Рис. 14. Содержимое папки проекта в Visual Studio Code

Откройте «**script.js»**. Инициализируйте сцену и камеру с помощью библиотеки «Three.js»:

```
const scene = new THREE.Scene();

const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);

camera.position.z = 5;
```

THREE.Scene() — это класс в библиотеке «Three.js», который представляет собой сцену в трехмерной графике. Сцена (Scene) в «Three.js» является контейнером, который содержит все объекты, светы, камеры и другие элементы. Позднее, нужно будет подключить библиотеку «Three.js» в файле «**index.html**» с html-кодом страницы.

Теперь нужно будет добавить рендерер «WebGL». «WebGL» (Web Graphics Library) — это JavaScript API, предназначенное для создания интерактивной 3D и 2D графики в веб-браузерах без использования плагинов. «WebGL» основан на OpenGL ES (OpenGL for Embedded Systems), что позволяет использовать аппаратное ускорение графики на компьютерах и мобильных устройствах.

Создадим экземпляр класса «**THREE.WebGLRenderer**», который представляет собой рендерер «WebGL», используемый для отображения трехмерной сцены в элементе «canvas» HTML:

```
const renderer = new THREE.WebGLRenderer({ canvas: document.getElementById('canvas') });
renderer.setSize(window.innerWidth, window.innerHeight);
```

Создадим новый источник света типа «**THREE.PointLight**»:

```
const light = new THREE.PointLight(0xffffff, 1);
```

Первый параметр «**0xffffff**» — это цвет света, указанный в формате RGB (в данном случае это белый свет). Второй параметр «1» — это интенсивность света, где «1» означает максимальную интенсивность.

```
light.position.set(10, 10, 10);
```

«**light.position.set**» – это позиция источника света в трехмерном пространстве. Здесь (10, 10, 10) задает координаты источника света в сцене. Это означает, что свет будет находиться в точке с координатами «x»=10, «y»=10, «z»=10.

Добавим источник света в сцену:

```
scene.add(light);
```

Загрузим наш файл ".glb" в сцену. Здесь, «**model.glb**» – имя файла нашей модели:

```
const loader = new THREE.GLTFLoader();
loader.load('model.glb', function (gltf) {
    scene.add(gltf.scene);
```

```
});
```

Добавим контроллер **«OrbitControls»**. Это позволит вращать камеру вокруг центра сцены с помощью мыши или сенсорного устройства. Он также позволяет изменять масштаб сцены, перемещать камеру и делать другие простые операции управления камерой:

```
const controls = new THREE.OrbitControls(camera, renderer.domElement);
```

Добавим обработчик события изменения размера окна браузера. Это позволит сцене корректно масштабироваться при изменении размера окна:

```
window.addEventListener('resize', function () {
	camera.aspect = window.innerWidth / window.innerHeight;
	camera.updateProjectionMatrix();
	renderer.setSize(window.innerWidth, window.innerHeight);
}, false);
```

Добавим функцию **«animate()»**, которая будет обеспечивать плавную анимацию путем обновления сцены и рендеринга каждый кадр:

```
function animate() {
    requestAnimationFrame(animate);
    renderer.render(scene, camera);
}
animate();
```

Функция «requestAnimationFrame()» предназначена для запуска анимации и вызывает функцию «animate()» перед отрисовкой следующего кадра. Это гарантирует, что анимация будет выполняться с частотой обновления экрана и не будет лишней нагрузки на процессор.

Теперь необходимо создать файл **«index.html»** в той же папке, где лежат файлы «script.js» и «model.glb», это будет наша веб-страница. Откройте **«index.html»**. Оформление может быть любым, вот пример простого оформления страницы:

```
<!DOCTYPE html>
<head>
	<meta charset="UTF-8">
	<meta name="viewport" content="width=device-width, initial-
scale=1.0"> <!-- Просмотр на мобильных устройствах -->

<title>3D Viewer</title>
```

```
      <style> <!-- CSS -->
         body, html {
             margin: 0;
             padding: 0;
             width: 100%;
             height: 100%;
             overflow: hidden;
         }
         #canvas-container {
             width: 100%;
             height: 100%;
             display: flex;
             justify-content: center;
             align-items: center;
             background-color: #000000;
         }
     </style>
</head>

<body>
    <div id="canvas-container">
        <canvas id="canvas"></canvas>
    </div>
<!-- Структура для отображения трехмерной сцены на веб-странице-->

     <script
src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js
"></script>
<!-- Подключение библиотеки Three.js к веб-странице -->

     <script
src="https://cdn.jsdelivr.net/npm/three/examples/js/loaders/GLTFLoader
.js"></script>
<!-- Подключение файла GLTFLoader.js из библиотеки Three.js, который
предоставляет загрузчик для файлов формата glTF (GL Transmission
Format). Этот загрузчик позволяет загружать и отображать 3D-модели в
формате glTF в проектах Three.js -->

     <script
src="https://cdn.jsdelivr.net/npm/three/examples/js/controls/OrbitCont
rols.js"></script>
<!-- Подключение файла OrbitControls.js из библиотеки Three.js, который
предоставляет удобные средства управления камерой с помощью мыши для
вращения, приближения и перемещения вокруг трехмерной сцены -->

     <script src="script.js"></script>
<!--Подключение JavaScript-файла script.js -->

</body>
```

```
</html>
```

При попытке открытия файла «index.html» и открытии кода разработчика, вы увидите ошибку, которая свидетельствует о том, что браузер блокирует доступ к ресурсам с локального файла из-за политики CORS (Cross-Origin Resource Sharing). Решить эту проблему можно разными способами. Воспользуемся Docker [8], для ознакомления с ним и наглядной демонстрации возможностей.

Docker — это платформа для разработки, доставки и запуска приложений в контейнерах. Он предоставляет средства для упаковки приложений и их зависимостей в контейнеры, которые могут быть развернуты на любой системе, поддерживающей Docker, без изменения окружения. Чтобы перенести контейнеры Docker на другой компьютер, вам нужно сначала сохранить контейнеры в образы, а затем перенести эти образы на другой компьютер.

Установите Docker с официального сайта [8].

Создайте папку «html» внутри папки вашего проекта и перенесите в нее «index.html», «script.js», «model.glb». Создайте текстовой файл внутри папки вашего проекта, там же, где лежит папка html с перенесённымив нее файлами. Напишите в текстовом файле следующие инструкции Dockerfile:

```
# Используем базовый образ с Apache HTTP Server
FROM httpd:latest
# Копируем файлы веб-приложения внутрь контейнера
COPY ./html /usr/local/apache2/htdocs/
```

Сохраните файл с именем «Dockerfile», **без расширения**, в папке вашего проекта.

Рис. 15. Содержимое папки проекта в проводнике

Теперь у вас есть «Dockerfile», который определяет инструкции для создания образа Docker. Будем использовать этот «Dockerfile» для создания образа Docker с помощью команды «docker build».

Откройте терминал или командную строку, перейдите в директорию, где находится ваш «Dockerfile». Например, если папка проекта с названием «Project» лежит на рабочем столе, используйте:

```
cd ~/Desktop/Project
```

Находясь в папке проекта, выполните следующую команду:

```
docker build -t my-apache-image .
```

Здесь «my-apache-image» — это имя образа Docker.

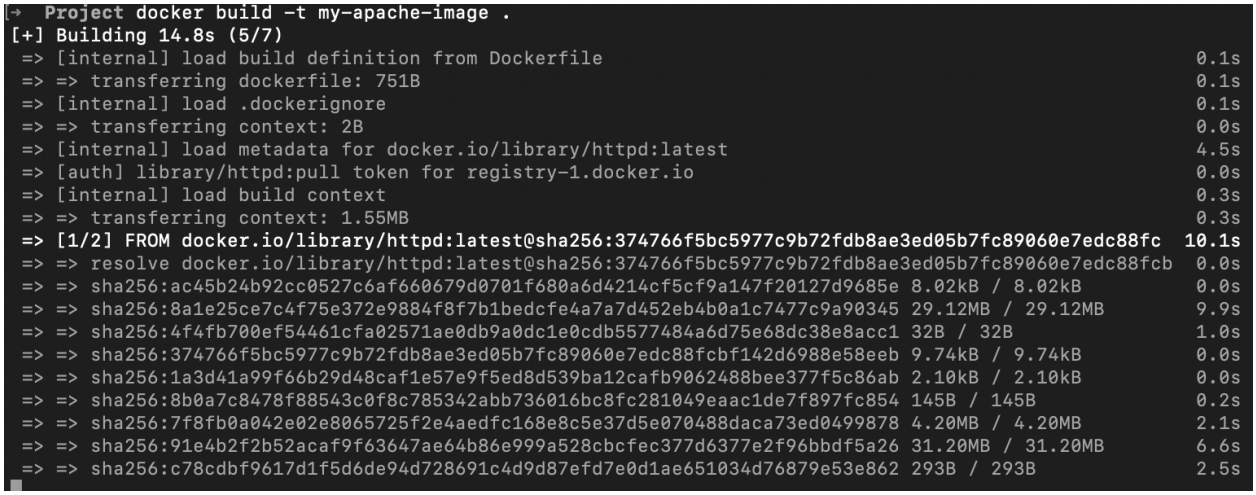

Рис. 16. Создание образа Docker

После этого будет создан образ Docker, включающий в себя локальный веб-сервер Apache и наше веб-приложение. Будем использовать этот образ для запуска контейнера Docker и просмотра веб-приложения локально.

После успешного создания образа Docker выполните команду для запуска контейнера:

```
docker run -d -p 8080:80 --name my-apache-container my-apache-image
```

«**-d**» означает запуск контейнера в фоновом режиме (daemon).

«**-p 8080:80**» пробрасывает порт 8080 на вашем хосте на порт 80 внутри контейнера, где запущен Apache HTTP Server.

«**--name my-apache-container**» задает имя контейнера. Можно использовать любое уникальное имя.

«**my-apache-image**» — это имя образа Docker.

После запуска контейнера, откроем и посмотрим наше веб-приложение локально. Для этого откройте веб-браузер и введите в адресной строке:

http://localhost:8080

Чтобы **остановить контейнер**, запустите команду **«docker ps»** в терминале, чтобы увидеть список запущенных контейнеров:

```
docker ps
```

Найдите контейнер, который вы хотите остановить, и скопируйте его CONTAINER ID. Затем выполните команду **«docker stop»**, указав CONTAINER ID вашего контейнера:

```
docker stop CONTAINER_ID
```

Для удаления контейнера воспользуйтесь следующей командой:

```
docker rm CONTAINER_ID
```

Для запуска контейнера после остановки используйте команду «docker start»:

```
docker start CONTAINER_ID
```