Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

ордена Трудового Красного Знамени

федеральное государственное

бюджетное образовательное учреждение

высшего образования

Московский Технический Университет Связи и Информатики



Кафедра “Математическая кибернетика и информационные технологии”

Отчет по учебной практике

по дисциплине “Объектно-ориентированное программирование”

на тему

“Web”

Выполнил студент

группы БВТ 2005

Макбари Д. А.

Проверил

Москва 2021

Оглавление

[1 Цель работы 3](#_Toc77656424)

[2 Постановка задачи 3](#_Toc77656425)

[3 Реализация задачи 1](#_Toc77656426)

[4 Пример работы 23](#_Toc77656427)

[5 Заключение 25](#_Toc77656428)

# **1 Цель работы**

Изучить html, css, js и сверстать себе страничку-портфолио.

# **2 Постановка задачи**

Сверстать себе страничку-портфолио.

# **3 Реализация задачи**

**HTML код:**

<!DOCTYPE html>

<html lang="ru">

<script src="libs/gl-matrix/dist/gl-matrix-min.js"></script>

<script src ="scripts/shaders.js"></script>

<script src ="scripts/matrixops.js"></script>

<script src="scripts/draw.js"></script>

<script src="scripts/animation.js"></script>

<head>

<meta charset="utf-8" />

<title>Портфолио</title>

</head>

<link rel="stylesheet" href="css/style.css">

<body>

<header>

<div class="myname"><a href="#animtext">Макбари Даниш Александрович</a></div>

<div class="menu">

<ul>

<li><a href="#me">Обо мне</a></li>

<li><a href="#works">Работы</a></li>

<li><a href="#contacts">Контакты</a></li>

</ul>

</div>

</header>

<div id="animtext" class="animtext">Привет! Меня зовут Даниш. Я учусь в МТУСИ.</div>

<div class="head" id="me">Обо мне</div>

<div class="text">Я yмею работать с HTML, CSS, JavaScript, Java, C/C++.</div>

<div class="head" id="works">Работы</div>

<table class="t3d">

<tr>

<td><canvas id="gouraudcube" width="200" height="200"></canvas></td>

<td><canvas id="gouraudsphere" width="200" height="200"></canvas></td>

<td><canvas id="phongcube" width="200" height="200"></canvas></td>

<td><canvas id="phongsphere" width="200" height="200"></canvas></td>

</tr>

</table>

<table class="tsettings">

<tr>

<td><p>Уровень тесселяции сферы:</p></td>

<td><input id="teschoice" type="range" min="0" max="6" step="1" value="3"></td>

</tr>

<tr>

<td><p>Позиция источника света(x, y, z):</p></td>

<td><input id="Cx" type="range" min="-15" max="15" step="1" value="10"></td>

<td><input id="Cy" type="range" min="-15" max="15" step="1" value="10"></td>

<td><input id="Cz" type="range" min="5" max="15" step="1" value="15"></td>

</tr>

<tr>

<td><p>Фоновая составляющая материала(r, g, b):</p></td>

<td><input id="Ar" type="range" min="0" max="1" step="0.1" value="0.3"></td>

<td><input id="Ag" type="range" min="0" max="1" step="0.1" value="0"></td>

<td><input id="Ab" type="range" min="0" max="1" step="0.1" value="0"></td>

</tr>

<tr>

<td><p>Диффузная составляющая материала(r, g, b):</p></td>

<td><input id="Dr" type="range" min="0" max="1" step="0.1" value="0.3"></td>

<td><input id="Dg" type="range" min="0" max="1" step="0.1" value="0"></td>

<td><input id="Db" type="range" min="0" max="1" step="0.1" value="0"></td>

</tr>

<tr>

<td><p>Зеркальная составляющая материала(r, g, b):</p></td>

<td><input id="Sr" type="range" min="0" max="1" step="0.1" value="0.6"></td>

<td><input id="Sg" type="range" min="0" max="1" step="0.1" value="0.7"></td>

<td><input id="Sb" type="range" min="0" max="1" step="0.1" value="0.7"></td>

</tr>

<tr>

<td><p>Коэффициент блeска:</p></td>

<td><p><input id="shininess" type="range" min="4" max="256" step="1" value="16"></p></td>

</tr>

</table>

<div class="text">Работа 1. Пример моей работы с HTML, CSS и JavaScript(WebGL2 и GLSL)</div>

<div class="text"></div>

<img class="center" src="images/clexample.png" width="768" height="432">

<div class="text">Работа 2. Пример моей работы с Си и OpenCL. Визуализация множества Мандельброта.</div>

<div class="text"></div>

<img class="center" src="images/desert.png" width="973" height="541">

<div class="text">Работа 3. Пример моей работы с Си, OpenGL4 и GLSL. Пустыня.</div>

<div class="head" id="contacts">Контакты</div>

<div class="text">Моя почта: <b>danish.m@mail.ru</b></div>

<div class="text">Мой номер: <b>8 (985) 906 38 64</b></div>

<div class="text">Мой телеграм: <b>t.me/danishmakbari</b></div>

<footer>

<div class="footertext">2021</div>

</footer>

</body>

</html>

**CSS код:**

body {

background-color: #FFFFFF;

}

\* {

margin: 0;

padding: 0;

}

header {

width: 100%;

height: 50px;

background: #1d1d1d;

position: fixed;

}

footer {

width: 100%;

height: 50px;

background: #1d1d1d;

}

.myname {

font-size: 25px;

text-align: left;

color: white;

margin-top: 10px;

margin-left: 20px;

}

.myname a {

text-decoration: none;

color: white;

}

.menu {

list-style: none;

position: absolute;

top: 10px;

right: 10px;

}

.menu li {

display: inline;

padding: 20px;

}

.menu a {

text-decoration: none;

color: white;

font-size: 20px;

}

.t3d {

margin: 0 auto;

padding-top: 50px;

}

.tsettings {

margin: 0 auto;

}

.head {

color: black;

font-size: 20px;

text-align: center;

padding-top: 75px;

font-weight: bold;

}

.text {

color: black;

font-size: 16px;

text-align: center;

padding-top: 30px;

padding-bottom: 20px;

}

.footertext {

color: white;

font-size: 16px;

text-align: center;

padding-top: 15px;

}

.animtext {

color:black;

font-size: 60px;

text-align: center;

padding-top: 350px;

padding-bottom: 450px

}

img.center {

display: block;

margin: 0 auto;

}

**JavaScript код:**

function printText(el) {

let letterTimeout = 20

let text = el.innerHTML

let i = 1

let print\_\_fn = function(){

if( i <= text.length ){

el.innerHTML = text.substr( 0, i );

setTimeout( arguments.callee, letterTimeout );

}

i++;

}

print\_\_fn()

};

octahedron = [

[ 0, 0, 0.5,

0.5, 0, 0,

0, 0.5, 0],

[-0.5, 0, 0,

0, 0, 0.5,

0, 0.5, 0],

[ 0.5, 0, 0,

0, 0, -0.5,

0, 0.5, 0],

[-0.5, 0, 0,

0, 0.5, 0,

0, 0, -0.5],

[ 0, 0, 0.5,

0, -0.5, 0,

0.5, 0, 0],

[-0.5, 0, 0,

0, -0.5, 0,

0, 0, 0.5],

[ 0.5, 0, 0,

0, -0.5, 0,

0, 0, -0.5],

[-0.5, 0, 0,

0, 0, -0.5,

0, -0.5, 0]

];

box\_vertices = [

-0.5, -0.5, -0.5, 0.0, 0.0, -1.0,

0.5, 0.5, -0.5, 0.0, 0.0, -1.0,

0.5, -0.5, -0.5, 0.0, 0.0, -1.0,

-0.5, 0.5, -0.5, 0.0, 0.0, -1.0,

0.5, 0.5, -0.5, 0.0, 0.0, -1.0,

-0.5, -0.5, -0.5, 0.0, 0.0, -1.0,

-0.5, -0.5, 0.5, 0.0, 0.0, 1.0,

0.5, -0.5, 0.5, 0.0, 0.0, 1.0,

0.5, 0.5, 0.5, 0.0, 0.0, 1.0,

0.5, 0.5, 0.5, 0.0, 0.0, 1.0,

-0.5, 0.5, 0.5, 0.0, 0.0, 1.0,

-0.5, -0.5, 0.5, 0.0, 0.0, 1.0,

-0.5, 0.5, 0.5, -1.0, 0.0, 0.0,

-0.5, 0.5, -0.5, -1.0, 0.0, 0.0,

-0.5, -0.5, -0.5, -1.0, 0.0, 0.0,

-0.5, -0.5, -0.5, -1.0, 0.0, 0.0,

-0.5, -0.5, 0.5, -1.0, 0.0, 0.0,

-0.5, 0.5, 0.5, -1.0, 0.0, 0.0,

0.5, 0.5, -0.5, 1.0, 0.0, 0.0,

0.5, 0.5, 0.5, 1.0, 0.0, 0.0,

0.5, -0.5, -0.5, 1.0, 0.0, 0.0,

0.5, -0.5, 0.5, 1.0, 0.0, 0.0,

0.5, -0.5, -0.5, 1.0, 0.0, 0.0,

0.5, 0.5, 0.5, 1.0, 0.0, 0.0,

-0.5, -0.5, -0.5, 0.0, -1.0, 0.0,

0.5, -0.5, -0.5, 0.0, -1.0, 0.0,

0.5, -0.5, 0.5, 0.0, -1.0, 0.0,

0.5, -0.5, 0.5, 0.0, -1.0, 0.0,

-0.5, -0.5, 0.5, 0.0, -1.0, 0.0,

-0.5, -0.5, -0.5, 0.0, -1.0, 0.0,

0.5, 0.5, -0.5, 0.0, 1.0, 0.0,

-0.5, 0.5, -0.5, 0.0, 1.0, 0.0,

0.5, 0.5, 0.5, 0.0, 1.0, 0.0,

-0.5, 0.5, 0.5, 0.0, 1.0, 0.0,

0.5, 0.5, 0.5, 0.0, 1.0, 0.0,

-0.5, 0.5, -0.5, 0.0, 1.0, 0.0

];

function draw(cdata, angle, scene)

{

let gl = cdata.gl;

let vao = cdata.vao;

let vbo = cdata.vbo;

let program = cdata.program;

let vertices = cdata.vertices;

let Lpos = glMatrix.vec3.create();

glMatrix.vec3.set(Lpos, scene.lightpos[0], scene.lightpos[1], scene.lightpos[2]);

let La = glMatrix.vec3.create();

glMatrix.vec3.set(La, 1.0, 1.0, 1.0);

let Ld = glMatrix.vec3.create();

glMatrix.vec3.set(Ld, 1.0, 1.0, 1.0);

let Ls = glMatrix.vec3.create();

glMatrix.vec3.set(Ls, 1.0, 1.0, 1.0);

let Ma = glMatrix.vec3.create();

glMatrix.vec3.set(Ma, scene.ambient[0], scene.ambient[1], scene.ambient[2]);

let Md = glMatrix.vec3.create();

glMatrix.vec3.set(Md, scene.diffuse[0], scene.diffuse[1], scene.diffuse[2]);

let Ms = glMatrix.vec3.create();

glMatrix.vec3.set(Ms, scene.specular[0], scene.specular[1], scene.specular[2]);

let shininess = scene.shininess;

let cameraPos = glMatrix.vec3.create();

glMatrix.vec3.set(cameraPos, 0.0, 0.0, 15.0);

let projMatrix = glMatrix.mat4.create();

glMatrix.mat4.identity(projMatrix);

glMatrix.mat4.perspective(projMatrix, 60.0 \* (180.0 / Math.PI), 1, 0.1, 100.0);

let viewMatrix = glMatrix.mat4.create();

glMatrix.mat4.identity(viewMatrix);

glMatrix.mat4.lookAt(viewMatrix, cameraPos, [0.0, 0.0, 0.0], [0.0, 1.0, 0.0]);

let modelMatrix = glMatrix.mat4.create();

glMatrix.mat4.identity(modelMatrix);

glMatrix.mat4.rotate(modelMatrix, modelMatrix, angle \* (180.0 / Math.PI), [1, 1, 0]);

glMatrix.mat4.scale(modelMatrix, modelMatrix, [5, 5, 5]);

gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

gl.useProgram(program);

setShaderProgramUniformMatrix4fv(gl, program, "mvp", getMVP(modelMatrix, viewMatrix, projMatrix));

setShaderProgramUniformMatrix4fv(gl, program, "modelMatrix", modelMatrix);

setShaderProgramUniformMatrix3fv(gl, program, "normalMatrix", getNormalMatrix(modelMatrix));

setShaderProgramUniform3fv(gl, program, "light.position", Lpos);

setShaderProgramUniform3fv(gl, program, "light.ambient", La);

setShaderProgramUniform3fv(gl, program, "light.diffuse", Ld);

setShaderProgramUniform3fv(gl, program, "light.specular", Ls);

setShaderProgramUniform3fv(gl, program, "material.ambient", Ma);

setShaderProgramUniform3fv(gl, program, "material.diffuse", Md);

setShaderProgramUniform3fv(gl, program, "material.specular", Ms);

setShaderProgramUniform1f(gl, program, "material.shininess", shininess);

setShaderProgramUniform3fv(gl, program, "cameraPos", cameraPos);

gl.bindVertexArray(vao);

gl.drawArrays(gl.TRIANGLES, 0, vertices);

}

function createCube(canvnam, vertshader, fragshader)

{

let canvas = document.querySelector(canvnam);

let gl = canvas.getContext("webgl2");

if (gl === null) {

alert("WebGL error. Can't create context.");

return;

}

gl.viewport(0, 0, canvas.width, canvas.height);

gl.clearColor(0xFF, 0xFF, 0xFF, 1.0);

gl.enable(gl.DEPTH\_TEST);

let vao = gl.createVertexArray();

let vbo = gl.createBuffer();

gl.bindVertexArray(vao);

gl.enableVertexAttribArray(0);

gl.enableVertexAttribArray(1);

gl.bindBuffer(gl.ARRAY\_BUFFER, vbo);

gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(box\_vertices), gl.STATIC\_DRAW);

gl.vertexAttribPointer(0, 3, gl.FLOAT, gl.FALSE, 6 \* Float32Array.BYTES\_PER\_ELEMENT, 0);

gl.vertexAttribPointer(1, 3, gl.FLOAT, gl.FALSE, 6 \* Float32Array.BYTES\_PER\_ELEMENT, 3 \* Float32Array.BYTES\_PER\_ELEMENT);

gl.bindVertexArray(null);

gl.bindBuffer(gl.ARRAY\_BUFFER, null);

let program = compileShaderProgram(gl, vertshader, fragshader);

let ret = {

'gl': gl,

'vao': vao,

'program': program,

'vertices': 36

};

return ret;

}

function subdivide(triangle, sphere, depth, sphere\_level)

{

if (depth == sphere\_level) {

for (let i = 0; i < 9; i += 3) {

let vec = glMatrix.vec3.create();

glMatrix.vec3.set(vec, triangle[i], triangle[i + 1], triangle[i + 2]);

glMatrix.vec3.normalize(vec, vec);

sphere.push(vec[0]);

sphere.push(vec[1]);

sphere.push(vec[2]);

}

return;

}

let triangle1 = [

(triangle[0] + triangle[3]) / 2, (triangle[1] + triangle[4]) / 2, (triangle[2] + triangle[5]) / 2,

(triangle[0] + triangle[6]) / 2, (triangle[1] + triangle[7]) / 2, (triangle[2] + triangle[8]) / 2,

triangle[0], triangle[1], triangle[2]

];

let triangle2 = [

(triangle[0] + triangle[6]) / 2, (triangle[1] + triangle[7]) / 2, (triangle[2] + triangle[8]) / 2,

(triangle[3] + triangle[6]) / 2, (triangle[4] + triangle[7]) / 2, (triangle[5] + triangle[8]) / 2,

triangle[6], triangle[7], triangle[8]

];

let triangle3 = [

(triangle[3] + triangle[6]) / 2, (triangle[4] + triangle[7]) / 2, (triangle[5] + triangle[8]) / 2,

(triangle[0] + triangle[3]) / 2, (triangle[1] + triangle[4]) / 2, (triangle[2] + triangle[5]) / 2,

triangle[3], triangle[4], triangle[5]

];

let triangle4 = [

(triangle[0] + triangle[3]) / 2, (triangle[1] + triangle[4]) / 2, (triangle[2] + triangle[5]) / 2,

(triangle[3] + triangle[6]) / 2, (triangle[4] + triangle[7]) / 2, (triangle[5] + triangle[8]) / 2,

(triangle[0] + triangle[6]) / 2, (triangle[1] + triangle[7]) / 2, (triangle[2] + triangle[8]) / 2

];

subdivide(triangle1, sphere, depth + 1, sphere\_level);

subdivide(triangle2, sphere, depth + 1, sphere\_level);

subdivide(triangle3, sphere, depth + 1, sphere\_level);

subdivide(triangle4, sphere, depth + 1, sphere\_level);

}

function genSphere(sphere\_level)

{

let sphere = new Array(0);

for (let i = 0; i < 8; i++)

subdivide(octahedron[i], sphere, 0, sphere\_level);

return sphere;

}

function createSphere(canvnam, vertshader, fragshader, sphere\_level)

{

let canvas = document.querySelector(canvnam);

let gl = canvas.getContext("webgl2");

if (gl === null) {

alert("WebGL error. Can't create context.");

return;

}

gl.viewport(0, 0, canvas.width, canvas.height);

gl.clearColor(0xFF, 0xFF, 0xFF, 1.0);

gl.enable(gl.DEPTH\_TEST);

let vao = gl.createVertexArray();

let vbo = gl.createBuffer();

gl.bindVertexArray(vao);

gl.enableVertexAttribArray(0);

gl.enableVertexAttribArray(1);

gl.bindBuffer(gl.ARRAY\_BUFFER, vbo);

let sphere = genSphere(sphere\_level);

gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(sphere), gl.STATIC\_DRAW);

gl.vertexAttribPointer(0, 3, gl.FLOAT, gl.FALSE, 0, 0);

gl.vertexAttribPointer(1, 3, gl.FLOAT, gl.FALSE, 0, 0);

gl.bindVertexArray(null);

gl.bindBuffer(gl.ARRAY\_BUFFER, null);

let program = compileShaderProgram(gl, vertshader, fragshader);

let ret = {

'gl': gl,

'vao': vao,

'vbo': vbo,

'program': program,

'vertices': 8 \* Math.pow(4, sphere\_level) \* 3

};

return ret;

}

function destroyData(data)

{

data.gl.deleteBuffer(data.vbo);

data.gl.deleteVertexArray(data.vao);

data.gl.deleteProgram(data.program);

}

window.onload =

function()

{

let data1 = createCube("#phongcube", phongVertexShader, phongFragmentShader);

let data2 = createCube("#gouraudcube", gouraudVertexShader, gouraudFragmentShader);

let data3 = createSphere("#phongsphere", phongVertexShader, phongFragmentShader, 3);

let data4 = createSphere("#gouraudsphere", gouraudVertexShader, gouraudFragmentShader, 3);

let angle1 = 0;

let angle2 = 45;

let teschoice = document.querySelector("#teschoice");

let prevval = teschoice.value;

let scene = {

'lightpos': [10, 10, 15],

'ambient': [0.2, 0, 0],

'diffuse': [0.3, 0, 0],

'specular': [0.7, 0.6, 0.6],

'shininess': 16

};

let lightpos;

let el = document.getElementById("animtext");

printText(el);

let intervalId = window.setInterval(

function()

{

angle1 += 0.0005;

angle2 += 0.0005;

if (angle1 >= 360) angle1 = 0;

if (angle2 >= 360) angle2 = 0;

scene.lightpos = [document.querySelector("#Cx").value,

document.querySelector("#Cy").value,

document.querySelector("#Cz").value];

scene.ambient = [document.querySelector("#Ar").value,

document.querySelector("#Ag").value,

document.querySelector("#Ab").value];

scene.diffuse = [document.querySelector("#Dr").value,

document.querySelector("#Dg").value,

document.querySelector("#Db").value];

scene.specular = [document.querySelector("#Sr").value,

document.querySelector("#Sg").value,

document.querySelector("#Sb").value];

scene.shininess = document.querySelector("#shininess").value;

if (prevval != teschoice.value) {

prevval = teschoice.value;

destroyData(data3);

destroyData(data4);

data3 = createSphere("#phongsphere", phongVertexShader, phongFragmentShader, prevval);

data4 = createSphere("#gouraudsphere", gouraudVertexShader, gouraudFragmentShader, prevval);

}

draw(data1, angle1, scene);

draw(data2, angle2, scene);

draw(data3, angle1, scene);

draw(data4, angle2, scene);

}

, 50);

};

function getNormalMatrix(modelMatrix)

{

let normalMatrix = glMatrix.mat4.create();

glMatrix.mat4.identity(normalMatrix);

glMatrix.mat4.invert(normalMatrix, modelMatrix);

glMatrix.mat4.transpose(normalMatrix, normalMatrix);

let nMat = glMatrix.mat3.create();

glMatrix.mat3.fromMat4(nMat, normalMatrix);

return nMat;

}

function getMVP(model, view, proj)

{

let mvp = glMatrix.mat4.create();

glMatrix.mat4.identity(mvp);

glMatrix.mat4.mul(mvp, proj, view);

glMatrix.mat4.mul(mvp, mvp, model);

return mvp;

}

phongVertexShader =

` #version 300 es

layout(location = 0) in vec3 vertexPosition;

layout(location = 1) in vec3 vertexNormal;

out vec3 fragPos;

out vec3 normalVec;

uniform mat4 mvp;

uniform mat4 modelMatrix;

uniform mat3 normalMatrix;

void main()

{

gl\_Position = mvp \* vec4(vertexPosition, 1);

normalVec = normalMatrix \* vertexNormal;

fragPos = vec3(modelMatrix \* vec4(vertexPosition, 1));

}

`;

phongFragmentShader =

` #version 300 es

precision mediump float;

in vec3 fragPos;

in vec3 normalVec;

layout(location = 0) out vec4 fragColor;

struct materialSettings {

vec3 ambient;

vec3 diffuse;

vec3 specular;

float shininess;

};

struct lightSettings {

vec3 position;

vec3 ambient;

vec3 diffuse;

vec3 specular;

};

uniform materialSettings material;

uniform lightSettings light;

uniform vec3 cameraPos;

vec3 ads(void)

{

vec3 normal = normalize(normalVec);

vec3 lightdir = normalize(light.position - fragPos);

vec3 viewdir = normalize(cameraPos - fragPos);

vec3 halfway = normalize(viewdir + lightdir);

vec3 ambient = light.ambient \* material.ambient;

vec3 diffuse = light.diffuse \* material.diffuse \* max(dot(lightdir, normal), 0.0);

vec3 specular = light.specular \* material.specular \* pow(max(dot(halfway, normal), 0.0), material.shininess);

return ambient + diffuse + specular;

}

void main()

{

vec3 ambient, diffuse, specular;

fragColor = vec4(ads(), 1);

}

`;

gouraudVertexShader =

` #version 300 es

precision mediump float;

layout(location = 0) in vec3 vertexPosition;

layout(location = 1) in vec3 vertexNormal;

out vec3 fragColor;

struct materialSettings {

vec3 ambient;

vec3 diffuse;

vec3 specular;

float shininess;

};

struct lightSettings {

vec3 position;

vec3 ambient;

vec3 diffuse;

vec3 specular;

};

uniform mat4 mvp;

uniform mat4 modelMatrix;

uniform mat3 normalMatrix;

uniform materialSettings material;

uniform lightSettings light;

uniform vec3 cameraPos;

vec3 ads(void)

{

vec3 normalVec = normalMatrix \* vertexNormal;

vec3 fragPos = vec3(modelMatrix \* vec4(vertexPosition, 1));

vec3 normal = normalize(normalVec);

vec3 lightdir = normalize(light.position - fragPos);

vec3 viewdir = normalize(cameraPos - fragPos);

vec3 halfway = normalize(viewdir + lightdir);

vec3 ambient = light.ambient \* material.ambient;

vec3 diffuse = light.diffuse \* material.diffuse \* max(dot(lightdir, normal), 0.0);

vec3 specular = light.specular \* material.specular \* pow(max(dot(halfway, normal), 0.0), material.shininess);

return ambient + diffuse + specular;

}

void main()

{

gl\_Position = mvp \* vec4(vertexPosition, 1);

fragColor = ads();

}

`;

gouraudFragmentShader =

` #version 300 es

precision mediump float;

in vec3 fragColor;

layout(location = 0) out vec4 glFragColor;

void main()

{

glFragColor = vec4(fragColor, 1);

}

`;

function compileShaderProgram(gl, vshadersrc, fshadersrc)

{

let vertexshader = compileShader(gl, gl.VERTEX\_SHADER, vshadersrc);

let fragmentshader = compileShader(gl, gl.FRAGMENT\_SHADER, fshadersrc);

let program = gl.createProgram();

gl.attachShader(program, vertexshader);

gl.attachShader(program, fragmentshader);

gl.linkProgram(program);

gl.deleteShader(vertexshader);

gl.deleteShader(fragmentshader);

return program;

}

function compileShader(gl, shadertype, src)

{

let shader = gl.createShader(shadertype);

gl.shaderSource(shader, src);

gl.compileShader(shader);

if (!gl.getShaderParameter(shader, gl.COMPILE\_STATUS)) {

alert(gl.getShaderInfoLog(shader));

gl.deleteShader(shader);

return null;

}

return shader;

}

function setShaderProgramUniform1f(gl, programID, name, value)

{

gl.uniform1f(gl.getUniformLocation(programID, name), value);

}

function setShaderProgramUniform1i(gl, programID, name, value)

{

gl.uniform1i(gl.getUniformLocation(programID, name), value);

}

function setShaderProgramUniform3fv(gl, programID, name, value)

{

gl.uniform3fv(gl.getUniformLocation(programID, name), value);

}

function setShaderProgramUniformMatrix3fv(gl, programID, name, value)

{

gl.uniformMatrix3fv(gl.getUniformLocation(programID, name), false, value);

}

function setShaderProgramUniformMatrix4fv(gl, programID, name, value)

{

gl.uniformMatrix4fv(gl.getUniformLocation(programID, name), false, value);

}

function webglHandleError(gl)

{

let err = gl.getError();

if (err != gl.NO\_ERROR) {

alert(err);

return;

}

}

# **4 Пример работы**

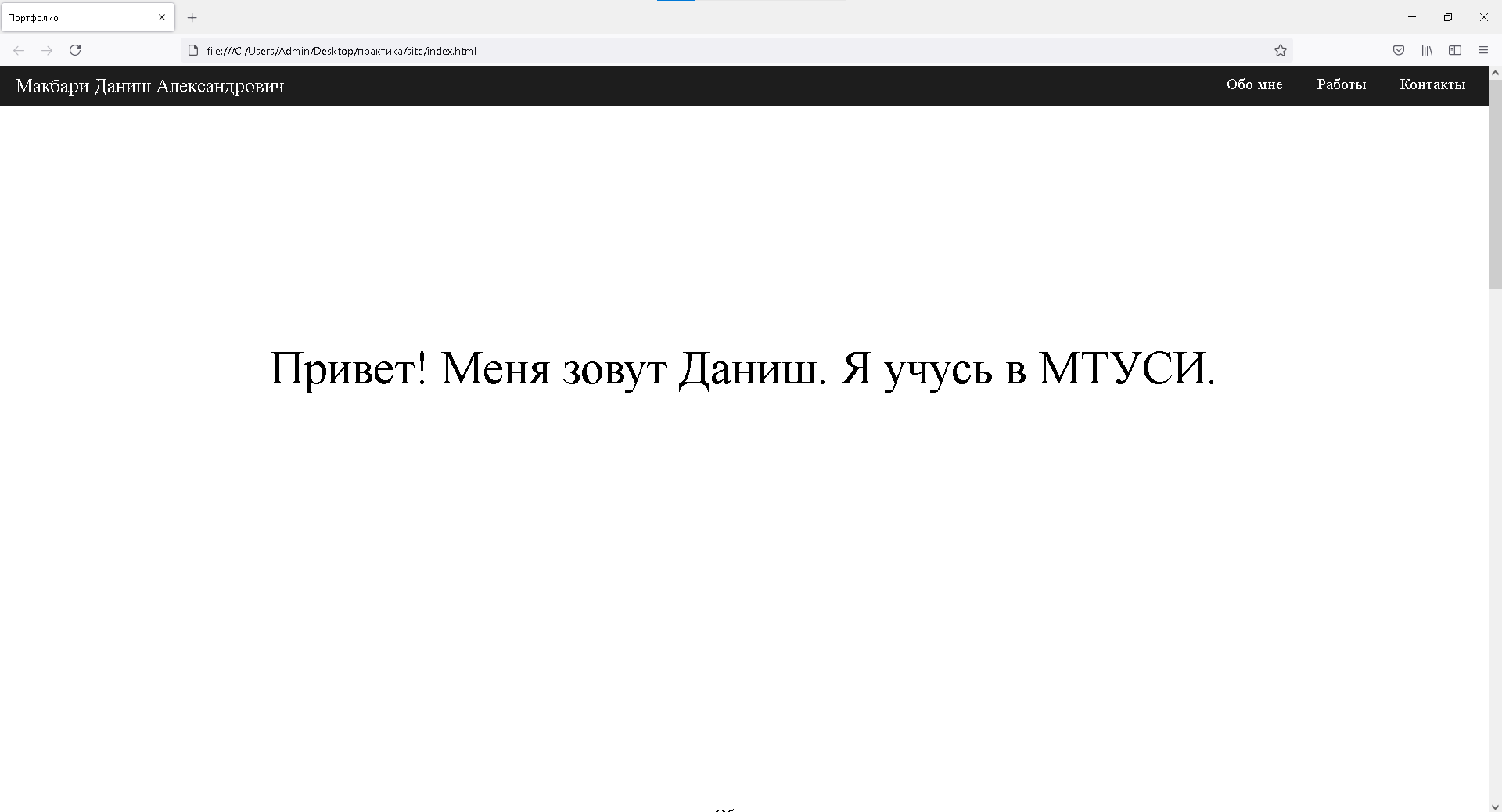


Рисунок 1

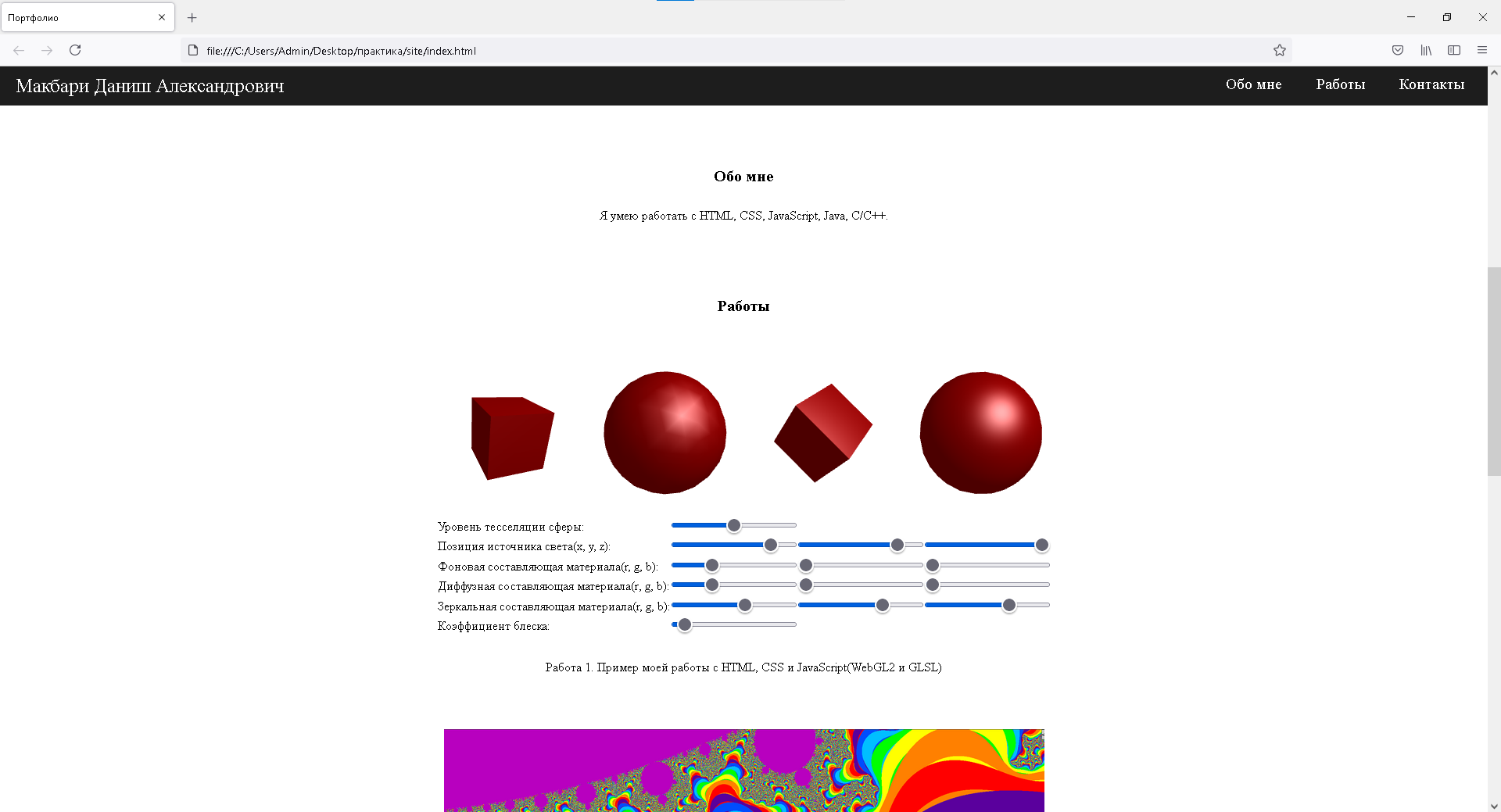


Рисунок 2

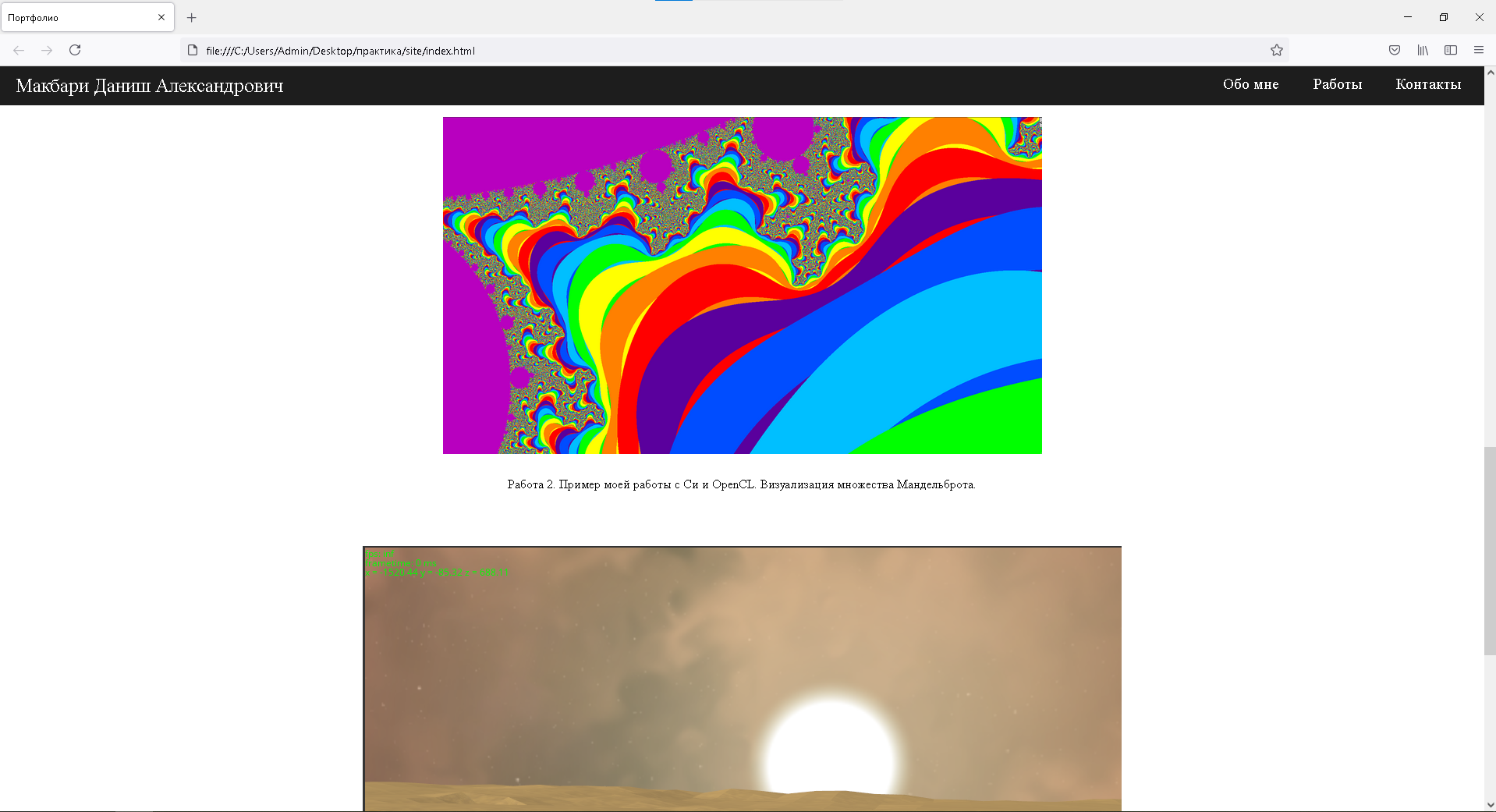


Рисунок 3

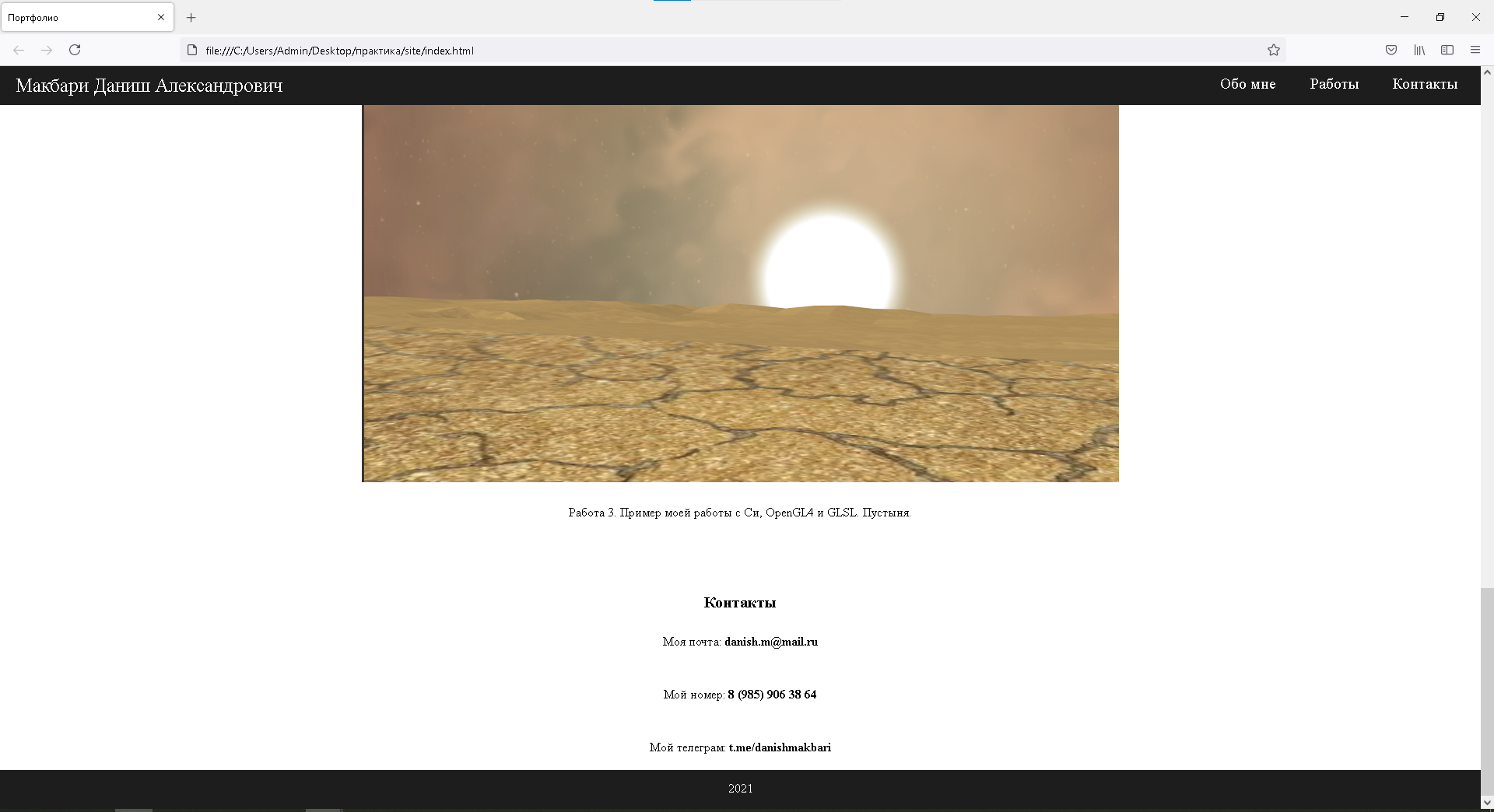


Рисунок 4

# **5 Заключение**

Изучил html, css, js и сверстал себе страничку-портфолио.