МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра цифрових технологій в енергетиці

**Візуалізація графічної та геометричної інформації**

Розрахунково графічна робота

**Виконали:**

Студент 5-го курсу, ІАТЕ,

гр. ТР-20мп

Огняник Д.М.

**Перевірив:** Демчишин А.А.

**Київ-2022**

**Завдання**

1. Нанести текстуру на поверхню з лабораторної роботи 2.
2. Реалізувати масштабування текстури (координати текстури), масштабування навколо визначеної користувачем точки.
3. Реалізувати можливість переміщати точку вздовж простору поверхні (u, v) за допомогою клавіатури: наприклад клавіші A і D переміщують точку вздовж параметра u, а клавіші W і S переміщують точку вздовж параметра v.

4. Завантажити код в репозиторій на GitHub

5. Створити гілку CGW, в яку помістити розроблений код.

6. Створити звіт до розрахунково-графічної роботи та завантажити в гілку CGW.

**Теоретична інформація**

WebGL дозволяє веб-контенту використовувати API, заснований на OpenGL ES 2.0, для візуалізації тривимірної графіки без використання плагінів у HTML елементі canvas у браузерах, які здійснюють його підтримку. WebGL програми складаються з коду керування, написаного на JavaScript і коду спеціальних ефектів (шейдерного коду), який виконується на графічному процесорі. WebGL елементи можуть бути змішані з іншими HTML-елементами та зібрані з іншими частинами веб-сторінки або тлом веб-сторінки.

Відображення текстури — це техніка визначення унікального кольору для кожного фрагмента, який утворює трикутник. Кольори походять із відображення. Для роботи з текстурою, яка має 2 координати, а поверхня 3 маємо перейти до UV координат. Змінюючи саме UV-координати можна досягти різних ефектів, таких як обертання, масштабування текстури, яка накладена на поверхню.

Для виконання розрахунково-графічної роботи необхідні знання про масштабування, перенесення об’єктів в WebGl.

Приклад реалізації для масштабування в роботі:

mat4 genScaleMat(vec3 v) {  
 return mat4(  
 vec4(v.x, 0.0, 0.0, 0.0),  
 vec4(0.0, v.y, 0.0, 0.0),  
 vec4(0.0, 0.0, v.z, 0.0),  
 vec4(0.0, 0.0, 0.0, 1.0)  
 );  
}

**Реалізація**

Для початку виконання графічно-розрахункової роботи було на основі другої лабораторної роботи створено гілку з назвою CGW, як вказано в завданні.

Далі необхідно додати декілька функцій для накладення текстури в main.js файлі. Розпочнемо з setTexture, яка необхідна для створення текстури і її встановлення, функція має вигляд:

function setTexture(gl, image) {  
 const texture = gl.createTexture();  
 gl.bindTexture(gl.TEXTURE\_2D, texture);  
 gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.LINEAR);  
 gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MAG\_FILTER, gl.LINEAR);  
 gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, image);  
}

В функцію передано зображення, яке необхідно спочатку найти на просторах інтернету та встановити посилання на його. Для цього створимо об’єкт типу Image та додамо посилання, так як нам потрібен час для того щоб зображення завантажилось, а JavaScript — асинхронна мова програмування, то додамо подію onLoad та будемо встановлювати текстуру за допомогою першого описанного метода:

const image = new Image();  
 image.src = "https://www.the3rdsequence.com/texturedb/download/257/texture/jpg/1024/green+moss-1024x1024.jpg";  
 image.crossOrigin = "anonymous";  
 image.onload = () => {  
 document.body.appendChild(image);  
 setTexture(gl, image);  
 }

Також створимо буфер для текстурних координат та прив’яжемо його до доданого атрибуту:

const tBuffer = gl.createBuffer();  
 gl.bindBuffer(gl.ARRAY\_BUFFER, tBuffer);  
 gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(texcoords), gl.STREAM\_DRAW);  
 gl.enableVertexAttribArray(shProgram.iAttribTexcoord);  
 gl.vertexAttribPointer(shProgram.iAttribTexcoord, 2, gl.FLOAT, false, 0, 0);

В Vertex шейдері додамо методи для масштабування

mat4 genScaleMat(vec3 v) {

return mat4(

vec4(v.x, 0.0, 0.0, 0.0),

vec4(0.0, v.y, 0.0, 0.0),

vec4(0.0, 0.0, v.z, 0.0),

vec4(0.0, 0.0, 0.0, 1.0)

);

}

Та метод трансформування:

mat4 genTranslateMat(vec3 v) {

return mat4(

vec4(1.0, 0.0, 0.0, v.x),

vec4(0.0, 1.0, 0.0, v.y),

vec4(0.0, 0.0, 1.0, v.z),

vec4(0.0, 0.0, 0.0, 1.0)

);

}

**Результати розрахунково-графічної роботи**

На рис.1 зображено результат виконання РГР з нанесенням текстури та початкової масштабованості.

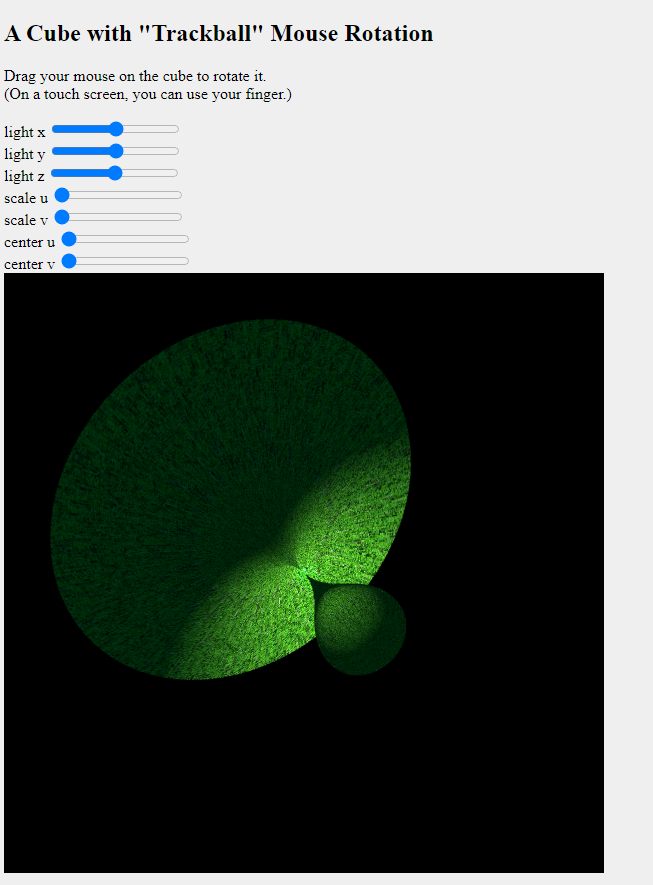
****

Рис. 1 --- Початкове положення поверхні

На рис. 2 було змінено scale та точку навколо якої ми працювали:

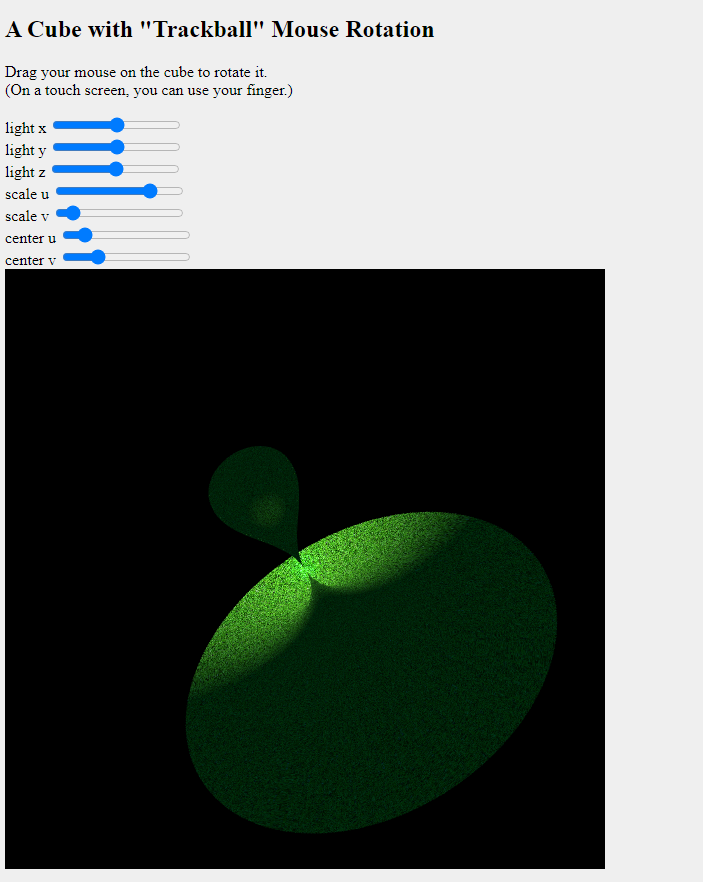
Т

Рис. 2 --- поверхня зі зміненими параметрами

**Код**

// Vertex shader

const vertexShaderSource = `

attribute vec3 vertex;

attribute vec2 texcoord;

uniform mat4 ModelViewProjectionMatrix;

uniform vec2 texScale;

uniform vec2 texCenter;

varying vec3 vVertex;

varying vec3 vVertexPosition;

varying vec2 vTexcoord;

mat4 genScaleMat(vec3 v) {

  return mat4(

    vec4(v.x, 0.0, 0.0, 0.0),

    vec4(0.0, v.y, 0.0, 0.0),

    vec4(0.0, 0.0, v.z, 0.0),

    vec4(0.0, 0.0, 0.0, 1.0)

  );

}

mat4 genTranslateMat(vec3 v) {

  return mat4(

    vec4(1.0, 0.0, 0.0, v.x),

    vec4(0.0, 1.0, 0.0, v.y),

    vec4(0.0, 0.0, 1.0, v.z),

    vec4(0.0, 0.0, 0.0, 1.0)

  );

}

vec2 scaleWithPoint(vec2 point, vec2 pivot, vec2 scale) {

    mat4 tran = genTranslateMat(vec3(pivot, 0.));

    mat4 tranBack = genTranslateMat(-vec3(pivot, 0.));

    mat4 scaleMat = genScaleMat(vec3(scale, 0.));

    vec4 point4 = vec4(point, 0., 0.);

    vec4 tr = tran \* point4;

    vec4 sc = tr \* scaleMat;

    vec4 trb = sc \* tranBack;

    return vec2(trb);

}

void main() {

    vec4 position = ModelViewProjectionMatrix \* vec4(vertex,1.0);

    vVertex = vertex;

    vVertexPosition = vec3(position) / position.w;

    vTexcoord = scaleWithPoint(texcoord, texCenter, texScale);

    gl\_Position = position;

}`;

// Fragment shader

const fragmentShaderSource = `

#ifdef GL\_FRAGMENT\_PRECISION\_HIGH

   precision highp float;

#else

   precision mediump float;

#endif

varying vec3 vVertex;

varying vec3 vVertexPosition;

varying vec2 vTexcoord;

uniform mat4 normalMatrix;

uniform vec3 lightPosition;

uniform sampler2D uTexture;

vec3 calculate\_light(vec3 position) {

    vec3 shape\_color = vec3(1., 1., 1.);

    vec3 light\_color = vec3(0., 1., 1.);

    vec3 ambient = light\_color \* 0.2;

    vec3 normal = normalize(vec3(normalMatrix \* vec4(vVertex, 0.)));

    vec3 light\_direction = normalize(lightPosition - position);

    float dot\_light = max(dot(normal, light\_direction), 0.0);

    vec3 diffuse = shape\_color \* dot\_light;

    float specular\_strength = 0.5;

    float spec = 0.;

    if (dot\_light > 0.) {

        vec3 view\_dir = normalize(-position);

        vec3 reflect\_dir = reflect(-light\_direction, normal);

        float spec\_angle = max(dot(view\_dir, reflect\_dir), 0.0);

        spec = pow(spec\_angle, 32.);

    }

    vec3 specular = specular\_strength \* spec \* light\_color;

    return ambient + diffuse + specular;

}

void main() {

    vec4 texture = texture2D(uTexture, vTexcoord);

    vec4 lightColor = vec4(calculate\_light(vVertexPosition), 1.);

    gl\_FragColor = texture \* lightColor;

}`;