Міністерство освіти й 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

Варіант №25

Виконав:

студент 5-го курсу

групи ТР-22мп ІАТЕ

Снитко О.Д.

Перевірив: Демчишин А.А.

Київ-2022

**Мета:** навчитись працювати з масштабуванням та обертанням текстури навколо початку координат та навколо вказаної користувачем точки.

**Завдання:** за варіантом розробити логіку масштабування та обертання текстури (парний - обертання, **непарний - масштабування**) відносно точки, яку обрав користувач.

Додати можливість управління точкою через клавіші a/d для v та w/s для u. Це все розробити на базі 2-ї практичної роботи, та завантажити на відповідну гілку в свій репозиторій. Також оформити звіт, який складається з: титульної сторінки, розділу з описом завдання (1 сторінка), розділом з описом теорії (1 сторінка), розділом з описом деталей впровадження (2 сторінки), розділ інструкції користувача зі скріншотами (2 сторінки), зразок вихідного коду (2 сторінки).

І при завершенні завантажити код та звіт на створену гілку.

**Теорія**

Основним завданням розрахунково-графічної роботи є: нанесення текстури на поверхню та масштабування текстури відносно точки на поверхні.

Нанесення текстури відображає розташування 2D-зображення на місце 3D-трикутника. WebGL використовує координати текстури для виконання нанесення текстури. Як і в багатьох інших аспектах графіки, координати текстури є відсотками, тобто значенням від 0 до 1. Позначення для координат текстури використовує (v,u) для представлення розташування зображення. Компонент u – це відсоток від ширини зображення, а компонент v – відсоток від висоти зображення. Кожній вершині моделі призначається координата текстури, і ці координати інтерполюються по всій поверхні трикутника, щоб визначити унікальне розташування на зображенні для кожного фрагмента трикутника. Тому обов’язково необхідно при генерації поверхні генерувати і uv координати у діапазоні від 0 до 1, які потім передати шейдеру для виконання процедури нанесення.

Масштабування текстури навколо точки відбувається за деяким алгоритмом: визначення центральної точки, потім необхідно сумістити центр текстури з початком координат за допомогою матриці translate.

На цьому етапі виконати сам процес масштабування відносно початку координат та повернути текстуру у обрану точку.

**Деталі впровадження**

Найперше, що необхідно було для виконання роботи - додати буфер для текстурних координат:

***shProgram***.iTexCoord = ***gl***.getAttribLocation(prog,"texcoord");

const tBuffer = ***gl***.createBuffer();

***gl***.bindBuffer(***gl***.*ARRAY\_BUFFER*, tBuffer);

***gl***.bufferData(***gl***.*ARRAY\_BUFFER*, new ***Float32Array***(uvs), ***gl***.*STREAM\_DRAW*);

***gl***.enableVertexAttribArray(***shProgram***.iTexCoord);

***gl***.vertexAttribPointer(***shProgram***.iTexCoord, 2, ***gl***.*FLOAT*, false, 0, 0);

Та відповідно заповнити його даними. Особливість координат текстур, що значення uv повинно бути в діапазоні від 0 до 1, тому стала необхідність нормування координат, це вирішилось в такий спосіб:

function normUv(b, z) {

return [b / 360, z / (2 \* l)];

}

, де поділили значення на максимальні, які були для цих параметрів.

Далі переходимо до завантаження текстури, поміщаємо в iTexCoord атрибут texcoord

***shProgram***.iTexCoord = ***gl***.getAttribLocation(prog, "texcoord");

Додаємо метод асинхронного завантаження текстури:

const teximage = new Image();

teximage.crossOrigin = "anonymous";

teximage.src = "https://phonoteka.org/uploads/posts/2021-06/1623129591\_14-phonoteka\_org-p-shagren-tekstura.jpg";

teximage.onload = function () {

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*, teximage);

draw();

}

після завантаження зображення встановлюємо текстуру.

Далі додамо в шейдери атрибути та varying зміні, які додавали в js файлі

У вертексний шейдер:

varying vec2 vTexcoord;

uniform vec2 texScale;

uniform vec2 texCenter;

attribute vec2 texcoord;

У фрагментний шейдер:

varying vec2 vTexcoord;

uniform sampler2D uTexture;

Додамо метод для масштабування відносно точки

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

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

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

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

vec4 trans = tran \* vec4(point, 0., 0.);

vec4 sc = trans \* scaleMat;

return vec2(sc \* tranBack);

}

Залишилось додати перехоплювачі натискання клавіш wasd для зміни uv параметрів з кроком 0.5:

***document***.onkeydown = (e) => {

if (e.key === "w") {

setCurrentValue(

***Vpoint***,

parseFloat(getCurrentValue(***Vpoint***)) + 0.5

)

} else if (e.key === "s") {

Інші клавіші обробляються схожим способом.

**Інструкція користувача зі скріншотами**

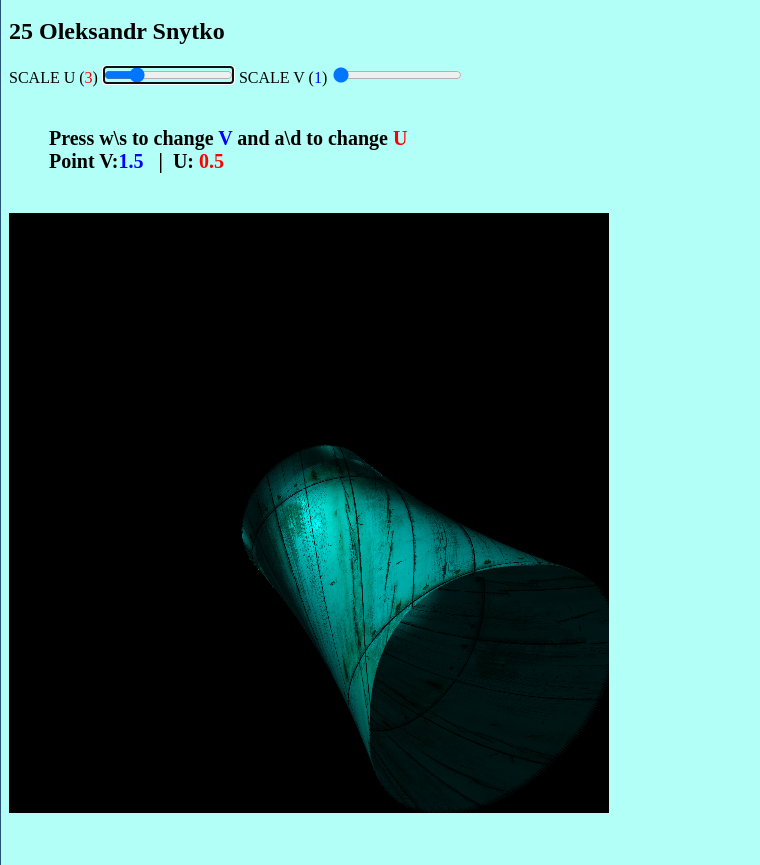
Відкривши сторінку в браузері можна побачити два інпути типу range, де можна обрати SCALE для U і V. Поруч виводиться їх значення. Далі йде інформація про керування точкою wasd і відповідно інформацію про місцезнаходження точки. Далі йде канвас з поверхнею.



За допомогою натискать на клавіші w+w+w+d отримаємо такий результат:



Та змінимо значення SCALE U на 3:



**Вихідний код**

Функція генерування поверхні в xyz та uv координатах:

function CreateSurfaceData() {

let vertices = [];

let uvs = [];

const l = 1;

const r1 = 0.5;

const r2 = 4;

function normUv(b, z) {

return [b / 360, z / (2 \* l)];

}

function r(a) {

a = deg2rad(a);

return (r2 - r1) \* ***Math***.pow(***Math***.sin((***Math***.PI \* a) / (4 \* l)), 2) \* 100 + r1;

}

for (let b = 0; b <= 360; b += 1) {

for (let a = 0; a <= 2 \* l; a += 0.1) {

const x = r(a) \* ***Math***.cos(deg2rad(b));

const y = r(a) \* ***Math***.sin(deg2rad(b));

const z = a;

vertices.push(x, y, z);

uvs.push(...normUv(b, a));

const a1 = a + 0.2;

const b1 = b + 5;

const x1 = r(a1) \* ***Math***.cos(deg2rad(b1));

const y1 = r(a1) \* ***Math***.sin(deg2rad(b1));

const z1 = a1;

vertices.push(x1, y1, z1);

uvs.push(...normUv(b1, a1));

}

}

return {vertices, uvs};

}

Головний метод вертексного шейдера:

void main() {

vec4 vertPos4 = ModelViewProjectionMatrix \* vec4(vertex, 1.0);

vec3 vertPos = vec3(vertPos4) / vertPos4.w;

vec3 normalInterp = vec3(normalMat \* vec4(vertex, 0.0));

gl\_Position = vertPos4;

vec3 normal = normalize(normalInterp);

vec3 lightDirection = normalize(lightPosition - vertPos);

//ambient

float ambientStrength = 0.1;

vec3 ambient = ambientStrength \* lightColor;

//diffuse

float diff = max(dot(normal, lightDirection), 0.0);

vec3 diffuse = diff \* lightColor;

//specular

float specularStrength = 0.5;

vec3 reflectDir = reflect(-lightDirection, normal);

float spec = pow(max(dot(lightDirection, reflectDir), 0.0), 32.0);

vec3 specular = specularStrength \* spec \* lightColor;

color = vec4(diffuse + ambient + specular, 1.0);

vTexcoord = scaleWithPoint(texcoord, texCenter, texScale);

}`;

Головний метод фрагментного шейдера:

void main() {

vec4 texture = texture2D(uTexture, vTexcoord);

gl\_FragColor = texture \* color;

}`;