**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО"**

Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота з дисципліни

“Візуалізація графічної та геометричної інформації”

на тему “Операції з текстурними координатами”

Виконав студент групи ТР-32мп  
Гасанов Ельдар Ігорович

Київ - 2023

1. **Завдання**

* Нанести текстуру на поверхню з практичного завдання №2.
* Реалізувати масштабування/обертання текстури (координати текстури) масштабування/обертання навколо визначеної користувачем точки – непарні варіанти реалізують масштабування, парні варіанти реалізують обертання.
* Розробити функціонал для переміщення точки 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.

1. **Теорія**

WebGL (Web Graphics Library) представляє собою JavaScript API, спрямоване на відтворення інтерактивної 2D- та 3D-графіки безпосередньо в контексті веб-браузера. Основна мета даного інструменту полягає в наданні можливості реалізації оптимізованих візуальних ефектів, таких як відображення текстур, масштабування, обертання та інтерактивне переміщення точок у веб-середовищі.

Однією з ключових особливостей WebGL є можливість використання шейдерів для визначення процесів обробки вершин та фрагментів графічних об'єктів. Шейдери надають можливість деталізованого керування, дозволяючи включати параметри, такі як текстурні координати та різноманітні перетворення.

Вершинні шейдери є ключовим етапом обробки графічних об'єктів у контексті відображення тривимірної моделі на екрані. Завданням вершинних шейдерів є отримання та обробка атрибутів, таких як просторове положення, кольорова інформація та координати текстури з вихідних даних вершин, з метою створення вихідного результату. Однією із поширених застосовуваних функцій вершинних шейдерів є застосування трансформацій до просторового положення вершин, таких як переміщення, обертання та масштабування, що є критичними для коректного розташування об'єктів у тривимірному просторі. Вихідні дані вершинного шейдера включають трансформоване положення вершини та інтерпольовані значення, які передаються фрагментному шейдеру.

Фрагментні шейдери, що іноді називають піксельними шейдерами, відповідають за обробку кожного пікселя, який буде відображено на екрані. Вони отримують інтерпольовані дані від вершинного шейдера, такі як координати кольору та текстури, а також інші параметри, включаючи інформацію про освітлення. Основним завданням фрагментного шейдера є визначення кінцевого кольору для кожного пікселя, що може включати в себе вибірку текстур, обчислення освітлення та інші ефекти. Кінцевий кольор виводу використовується для відображення конкретного пікселя на екрані.

Для досягнення високої якості візуалізації, WebGL використовує текстурні об'єкти, які створюються та прив'язуються до конкретних текстурних одиниць, що дозволяє ефективно вибирати і застосовувати їх у шейдерах.

Матричні перетворення використовуються для управління положенням, масштабуванням і обертанням 3D-моделей, надаючи можливість точного розміщення графічних об'єктів у віртуальному просторі. Цей набір функцій робить WebGL потужним інструментом для створення вражаючих інтерактивних веб-графіків та візуалізацій.

Відображення текстури є методологією в галузі комп'ютерної графіки, спрямованою на реалізацію реалістичних поверхонь шляхом накладання зображень або текстур на 3D-моделі. У контексті WebGL процес відображення текстур включає асоціацію кожної вершини 3D-об'єкта з координатами текстури (u, v), які в подальшому використовуються для вибірки кольорів з зображення текстури. Цей підхід сприяє покращенню візуального вигляду об'єкта, надаючи деталізовану інформацію про його поверхню.

Координати текстури (u, v) представляють собою ключові параметри, що визначають спосіб нанесення текстури на поверхню. Значення цих координат знаходяться в межах від 0 до 1 і використовуються для точного вибору пікселів на зображенні текстури. У контексті WebGL координати текстури пов'язані з кожною вершиною 3D-моделі та піддаються інтерполяції по всій поверхні під час візуалізації. Це гарантує, що текстура акуратно адаптується до геометрії моделі, забезпечуючи точне відтворення на поверхні об'єкта.

1. **Деталі розробки**

За варіантом було надано “Clover knot”. При виконанні практичного завдання №2 було розроблено програму, що виводить поверхню у вигляді суцільних трикутників.

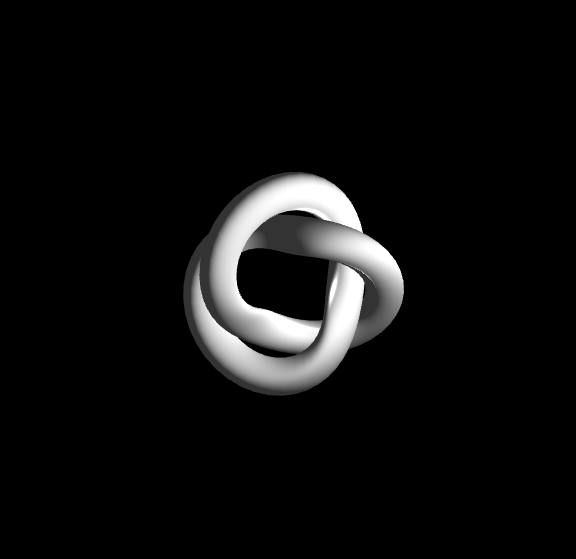


Рисунок 1 – вигляд заданої поверхні

Для виконання розрахунково-графічної роботи було обрано наступне зображення.



Рисунок 2 – обрана тектура

Текстуру було накладено на поверхню. Для цбого спочатко було підготовано буфер текстурних кординат, кожна з яких відповідає елементам масиву з буферу вершин. Згідно варіанту текстура має масштабуватися, тому було створено відповідний uniform, який використовується для визначення фактору масштабування.

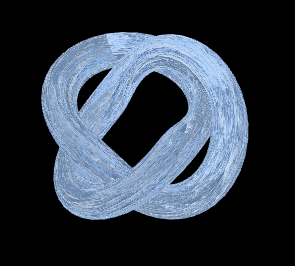


Рисунок 3 – фігура з накладеною текстурою

Для відображення точки відносно якої буде здійснюватись трансформація текстури – було створено об’єкт класу Model. Цей об’єкт відображається як сфера на поверхні.

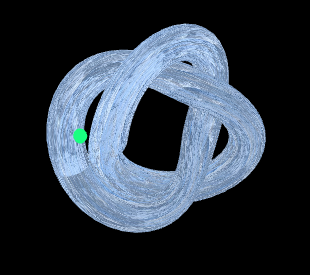
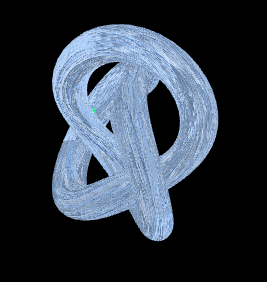
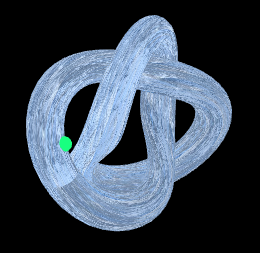


Рисунок 4 – точка на текстурованій поверхні

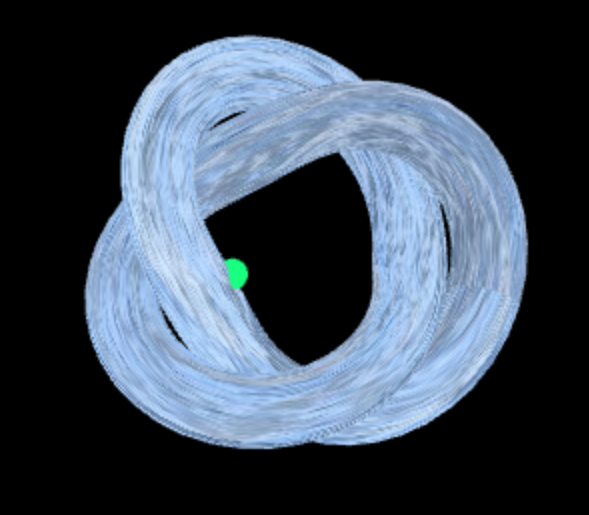
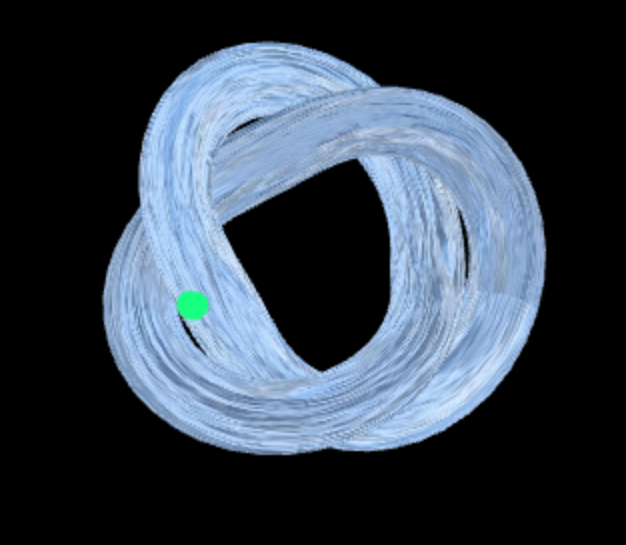
1. **Інструкції користувача**

Фігуру можна обертати відносно центру затиснувши ліву клавішу миші та потягнувши в сторону бажаного обертання.



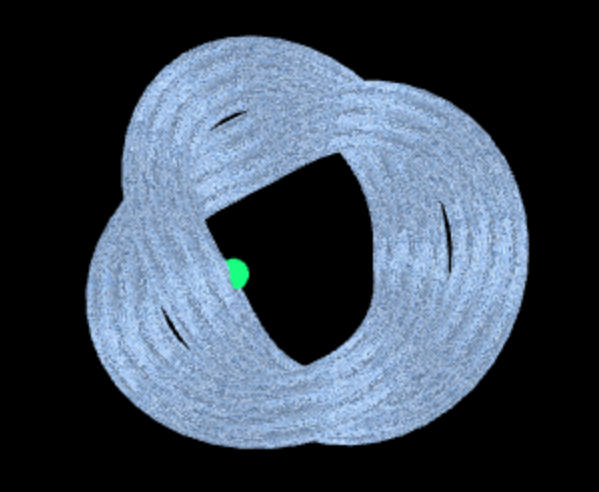
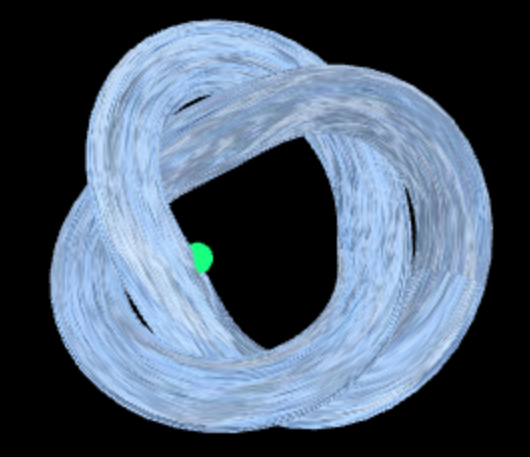
Рисунки 5, 6 – фігура до та після обертання

Переміщувати точку відносно якої здійснюється обертання можна за допомогою клавіш WASD. Кожне натискання переміщує точку по поверхні на визначений крок. Переміщення здійснюється до визначеної межі.



Рисунки 7, 8 – фігура з різними положеннями об’єкту Model

За допомогою слайдера можна змінювати фактор масштабування відносно умовної точки на поверхні. Можна помітити, що при переміщенні точки зміщується і текстура, аджеу масштабування відбувається відносно іншої точки на поверхні , яка в свою чергу відповідає іншій текстурній координаті.



Рисунки 9, 10 - Демонстрація зміни фактору масштабування текстури

1. **Код програми**

function draw() {

    gl.clearColor(0, 0, 0, 1);

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

    /\* Set the values of the projection transformation \*/

    let projection = m4.perspective(Math.PI / 8, 1, 8, 12);

    /\* Get the view matrix from the SimpleRotator object.\*/

    let modelView = spaceball.getViewMatrix();

    let rotateToPointZero = m4.axisRotation([0.707, 0.707, 0], 0.7);

    let translateToPointZero = m4.translation(0, 0, -10);

    let matAccum0 = m4.multiply(rotateToPointZero, modelView);

    let matAccum1 = m4.multiply(translateToPointZero, matAccum0);

    /\* Multiply the projection matrix times the modelview matrix to give the

       combined transformation matrix, and send that to the shader program. \*/

    let modelViewProjection = m4.multiply(projection, matAccum1);

    gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, modelViewProjection);

    let normalMatrix = m4.identity();

    m4.inverse(modelView, normalMatrix);

    m4.transpose(normalMatrix, normalMatrix);

    gl.uniformMatrix4fv(shProgram.iNormalMatrix, false, normalMatrix);

    /\* Draw the six faces of a cube, with different colors. \*/

    gl.uniform4fv(shProgram.iColor, [0.1, 1, 0.5, 1]);

    gl.uniform1f(shProgram.iTime, (Date.now() - timestamp) \* 0.001);

    gl.uniform2fv(shProgram.iPointOfScaling, [map(pointOfScaling[0], 0, Math.PI \* 12, 0, 1), map(pointOfScaling[1], 0, Math.PI \* 2, 0, 1)]);

    gl.uniform3fv(shProgram.iPointPosition, cloverKnot(pointOfScaling[0], pointOfScaling[1], 2, 0.5));

    gl.uniform1f(shProgram.iScalingNumber, parseFloat(document.getElementById('scale').value));

    surface.Draw();

    gl.uniform1i(shProgram.iTranslateLight, true);

    lightSurf.Draw();

    gl.uniform1i(shProgram.iTranslateLight, false);

}

void main() {

vec4 texture1 = translation(vec3(-pointOfScaling,0))\*vec4(texture,0,1);

vec4 texture2 = scaling(scalingNumber)\*texture1;

vec4 texture3 = translation(vec3(pointOfScaling,0))\*texture2;

vTexture = vec2(texture3.x,texture3.y);

vNormal = vec3(NormalMatrix \* vec4(normal, 0.0));

vVertex = vertex;

gl\_Position = ModelViewProjectionMatrix \* vec4(vertex,1.0);

if( translateLight ){

gl\_Position = ModelViewProjectionMatrix\*translation(pointPos)\* vec4(vertex,1.0);

//gl\_Position = ModelViewProjectionMatrix\* vec4(vertex,1.0);

}

}`;