**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”

Варіант 10

Виконав студент групи ТР-31мп

Ізварін Євген Ігорович

Київ - 2024

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

WebGL представляє собою JavaScript API для відтворення інтерактивної 2D- і 3D-графіки у веб-браузері. Для здійснення відображення текстур, масштабування, обертання та інтерактивного переміщення точок, WebGL API має функціонал для обробки шейдерів, текстур і матричних перетворень.

* Шейдери використовуються для визначення того, як обробляються вершини та фрагменти, надаючи можливість включати текстурні координати та різні перетворення.
* Текстурні об'єкти створюються та асоціюються з конкретними текстурними одиницями для використання в шейдерах.
* Матричні перетворення застосовуються для управління положенням, масштабуванням і обертанням 3D-моделі.

Шейдери WebGL

У WebGL шейдери грають важливу роль у конвеєрі візуалізації, забезпечуючи гнучкий і програмований підхід до визначення вигляду 3D-графіки. В цій технології існують два основних типи шейдерів: вершинні та фрагментні.

Вершинні шейдери відповідають за обробку кожної вершини 3D-моделі перед її відображенням на екрані. Вони отримують атрибути з даних вершин, таких як положення, колір і координати текстури, і обробляють ці дані для створення виходу. Один з часто використовуваних методів у вершинних шейдерах - застосування трансформацій до позицій вершин, таких як переміщення, обертання та масштабування, що є важливими для розташування об’єктів у 3D-просторі. Вихідні дані вершинного шейдера містять трансформовану позицію вершини та інтерпольовані значення, які передаються фрагментному шейдеру.

Фрагментні шейдери, іноді називані піксельними, працюють з кожним пікселем, який буде відображений на екрані. Вони отримують інтерпольовані значення від вершинного шейдера, такі як координати кольору та текстури, а також інші дані, наприклад, про освітлення. Головне завдання фрагментного шейдера - визначити остаточний колір кожного пікселя, що може включати вибірку текстур, обчислення освітлення та інші ефекти. Остаточний колір використовується для малювання пікселя на екрані.

Обидва типи шейдерів, вершинні та фрагментні, написані мовою GLSL (OpenGL Shading Language) і повинні бути скомпільовані перед використанням. Після компіляції вони об'єднуються в програму шейдера, яку потрібно зв'язати перед відтворенням, щоб WebGL використовував визначені вершинні та фрагментні шейдери у конвеєрі відтворення.

Uniforms - це значення, які залишаються незмінними для всіх вершин або фрагментів примітиву під час візуалізації. Вони дозволяють передавати зовнішні дані шейдерам, такі як матриці трансформації або інформація про глобальне освітлення.

Attributes - це дані для кожної вершини, які різняться між вершинами. Вони використовуються для передачі інформації, такої як положення вершин, нормалі та координати текстури.

Координати текстури зазвичай передаються як атрибути до вершинного шейдера, а потім інтерполюються через примітив для використання у фрагментному шейдері. У фрагментному шейдері координати текстури використовуються для вибірки кольорів із текстур, які потім визначають остаточний колір кожного пікселя.

Накладання текстур. Текстурні координати

Відтворення текстури - це техніка у галузі комп'ютерної графіки, яка дозволяє створювати реалістичні поверхні шляхом застосування зображень або текстур на 3D-моделі. У випадку WebGL відтворення текстур включає прив'язку кожної вершини 3D-об'єкта до координат текстури (u, v), які потім використовуються для вибірки кольорів із зображення текстури. Це покращує зовнішній вигляд об'єкта, надаючи деталізовану інформацію про його поверхню.

Координати текстури (u, v) є важливими параметрами, які визначають метод застосування текстури на поверхню. Вони розташовані від 0 до 1 і використовуються для звертання до конкретних точок на зображенні текстури. У випадку WebGL координати текстури пов'язані з кожною вершиною 3D-моделі та інтерполюються по всій поверхні під час візуалізації. Це гарантує, що текстура точно адаптується до моделі.

1. **Деталі розробки**

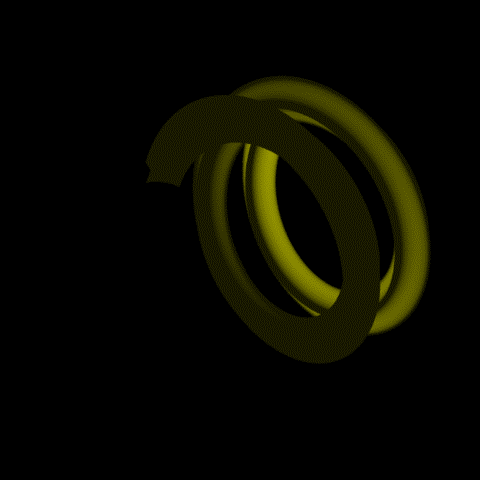
За варіантом мені було надано astroid helicoid. При виконанні практичного завдання №2 було розроблено програму, що виводить поверхню у вигляді суцільних трикутників.

Рисунок 1 – Вигляд поверхні

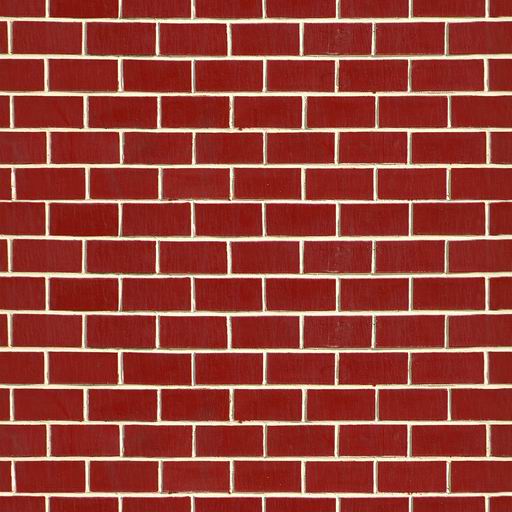
Я обрав наступне зображення для подальшого виконання розрахунково-графічної роботи. Для підтримки на більшій кількості операційних систем та браузерів було обрано зображення розміром 512х512 пікселів. Формат зображення .jpg

Рисунок 2 – Зображення текстури

Було накладено текстуру на поверхню. Для накладання текстури на поверхню було підготовлено буфер текстурних координат, кожна з яких відповідає елементу масива з буферу вершин. Згідно варіанту текстура має обертатися тому було створено відповідний uniform, який визначатиме кут обертання. В програмі шейдера для визначення кольору пікселя фігури на яку накладено текстуру необхідно використовувати функцію texture2D().

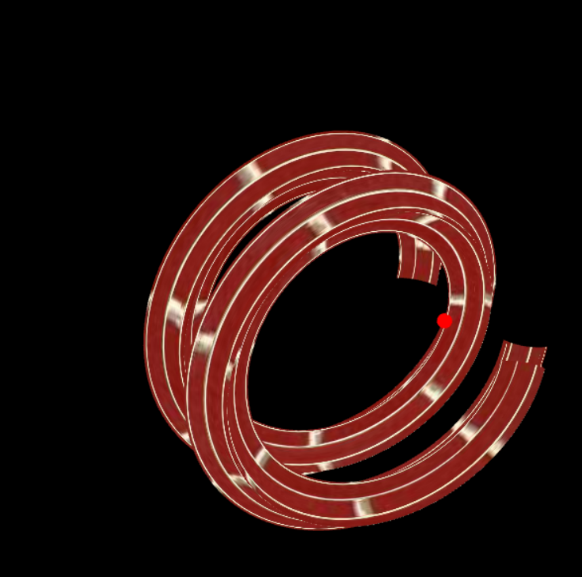
Ця функція першим аргументом приймає обʼєкт класу sampler2D, який зберігає дані про зображення, а другим текстурну координату.

Рисунок 3 – текстура накладена на поверхню

Було створено новий обʼєкт класу Model для відображення точки відносно якої буде здійснюватись трансформація текстури. Обʼєкт графічно відображається як сфера. Сфера перебуває на поверхні.

Рисунок 4 – точка на текстурі поверхні

Згідно варіанту було імплементовано обертання текстури відносно точки на поверхні.

1. **Інструкції користувача**

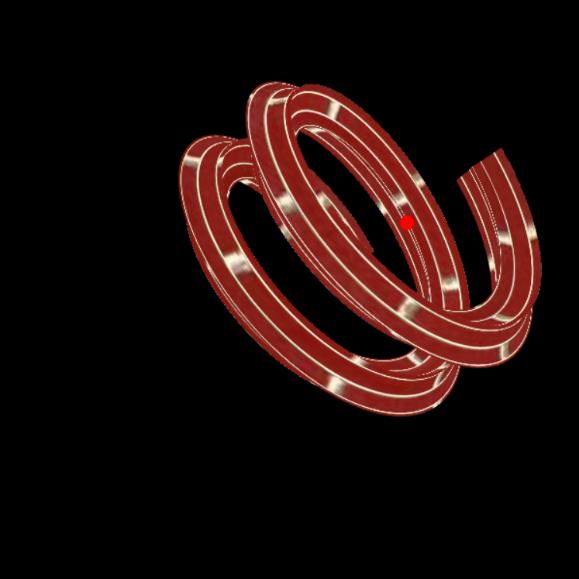
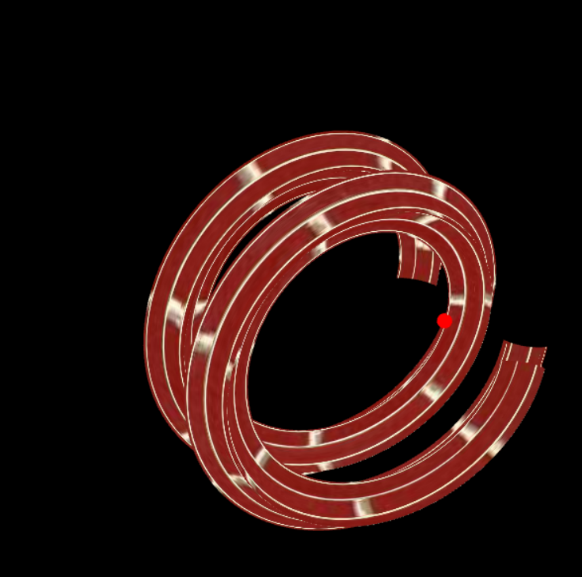
Фігуру можна обертати відносно центру затиснувши ліву клавішу миші та потягнувши в сторону бажаного обертання.

Рисунок 5 - Вигляд фігури до та після обертання

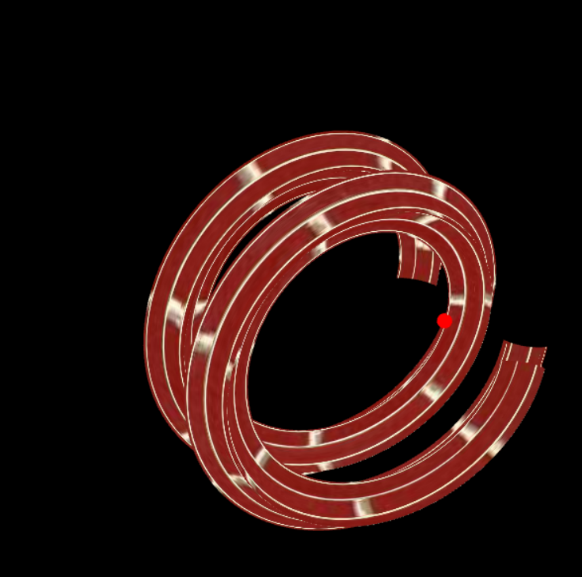
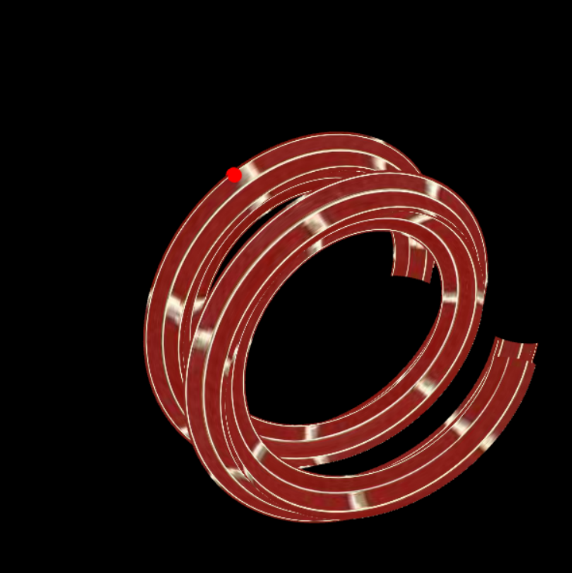
Переміщувати точку відносно якої здійснюється обертання можна за допомогою клавіш WASD. Кожне натискання переміщує точку по поверхні на визначений крок. Переміщення здійснюється до визначеної межі.

Рисунок 6 – демонстрація переміщення точки відносно нерухомо фігури

За допомогою слайдера з підписом “Angle” можна змінювати кут обертання відносно умовної точки на поверхні. Коли значення кута обертання не нулове можна помітити, що при переміщенні точки зміщується і текстура, адже обертання відбувається відносно іншої точки на поверхні , яка в свою чергу відповідає іншій текстурній координаті.

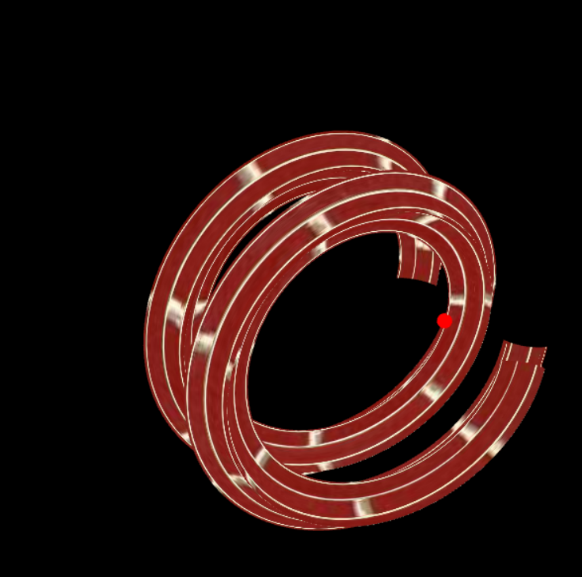


Рисунок 7 - Демонстрація зміни кута обертання текстури

При перезавантаженні сторінки обертання фігури, а також значення кута обертання буде встановлено по замовчуванню. Положення точки відносно поверхні також буде скинуто до значення по замовчуванню.

1. **Код програми**

Код функції частини програми на javascript

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);

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

let col = hexToRgb(document.getElementById('col').value)

let z = document.getElementById('z').value

let r = document.getElementById('r').value

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

gl.uniform3fv(shProgram.iLightPos, [r \* Math.sin(Date.now() \* 0.001), r \* Math.cos(Date.now() \* 0.001), z]);

gl.uniform2fv(shProgram.iTextu, textu);

gl.uniform1f(shProgram.iAngle, document.getElementById('angle').value);

surface.Draw();

gl.uniform3fv(shProgram.iLightPos, [1000, Math.cos(Date.now() \* 0.001), 0]);

gl.uniformMatrix4fv(shProgram.iModelViewProjectionMatrix, false, m4.multiply(modelViewProjection,

m4.translation(...ahv(textu[0]\*Math.PI\*4,textu[1]\*Math.PI\*2))));

light.Draw();

}

Код функції частини програми на GLSL

void main() {

vec3 N = normalize(var\_Normal);

vec3 L = normalize(lightPos - var\_Vertex);

// Lambert's cosine law

float lambertian = max(dot(N, L), 0.0);

float specular = 0.0;

if(lambertian > 0.0) {

vec3 R = reflect(-L, N); // Reflected light vector

vec3 V = normalize(-var\_Vertex); // Vector to viewer

// Compute the specular term

float specAngle = max(dot(R, V), 0.0);

specular = pow(specAngle, 50.0);

}

vec3 col = color.rgb\*0.1+lambertian\*color.rgb+vec3(1.0)\*specular;

gl\_FragColor = vec4(col,1.0);

gl\_FragColor = texture2D(tmu,var\_Texture);

if(lightPos.x>100.0){

gl\_FragColor = vec4(1.0,0.,0.,1.0);

}

}`;