**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 4

Виконав студент групи ТР-32мп

Васильцов Кирило Дмитрович

Київ - 2023

**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.

**Теорія**

WebGL

WebGL - це JavaScript API для створення інтерактивної 2D та 3D графіки у веб-браузерах. Для досягнення ефектів, таких як відображення текстур, масштабування, обертання та інтерактивне переміщення точок, WebGL надає функції для обробки шейдерів, текстур і матричних перетворень.

Шейдери використовуються для визначення того, як обробляються вершини та фрагменти, що дозволяє включати текстурні координати та здійснювати різні перетворення.

Текстурні об'єкти створюються та прив'язуються до конкретних текстурних одиниць для вибірки в шейдерах.

Матричні перетворення застосовуються для управління положенням, масштабуванням і обертанням 3D-моделі.

Шейдери WebGL

WebGL відіграє важливу роль у візуалізації, забезпечуючи гнучкий та програмований підхід до визначення вигляду 3D-графіки. У цьому контексті шейдери є ключовим елементом, поділяючись на два основних типи: вершинні та фрагментні.

Вершинні шейдери відповідають за обробку кожної вершини 3D-моделі перед її відображенням. Вони використовують атрибути, такі як положення, колір і координати текстури, для створення виходу. Застосування трансформацій, таких як переміщення, обертання та масштабування, є загальним використанням вершинних шейдерів, впливаючи на позицію об'єктів у 3D-просторі. Вихідні дані містять перетворену позицію вершини та інтерпольовані значення для передачі фрагментному шейдеру.

Фрагментні шейдери (іноді піксельні) працюють з кожним пікселем, що буде відображений на екрані. Вони отримують інтерпольовані значення від вершинного шейдера, такі як координати кольору та текстури, а також інші дані освітлення. Основне завдання фрагментного шейдера - визначити кінцевий колір кожного пікселя, враховуючи текстури та освітлення.

Обидва типи шейдерів пишуться мовою GLSL (OpenGL Shading Language) та вимагають компіляції перед використанням. Після цього вони об'єднуються в програму шейдера, яку потрібно зв'язати перед відтворенням для використання в конвеєрі.

Uniforms - це константні значення для всіх вершин чи фрагментів під час візуалізації. Вони передають зовнішні дані, такі як матриці трансформації, шейдерам.

Attributes - це дані, що відрізняються між вершинами. Вони використовуються для передачі інформації, такої як положення, нормали та координати текстури.

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

Накладання текстур. Текстурні координати

Відображення текстур - це метод у комп'ютерній графіці, що дозволяє створювати реалістичні поверхні шляхом застосування зображень або текстур на 3D-моделі. У контексті WebGL відображення текстур включає асоціювання кожної вершини 3D-об'єкта з координатами текстури (u, v), які використовуються для вибору кольорів із текстурного зображення. Це значно підвищує візуальну якість об'єкта, надаючи деталізовану інформацію про його поверхню.

Координати текстури (u, v) є ключовими параметрами, що визначають спосіб нанесення текстури на поверхню. Значення цих координат знаходяться в межах від 0 до 1 і використовуються для визначення конкретних точок на текстурному зображенні. У контексті WebGL координати текстури пов'язані з кожною вершиною 3D-моделі та автоматично інтерполюються по всій поверхні під час візуалізації. Це гарантує точне відображення текстури на моделі

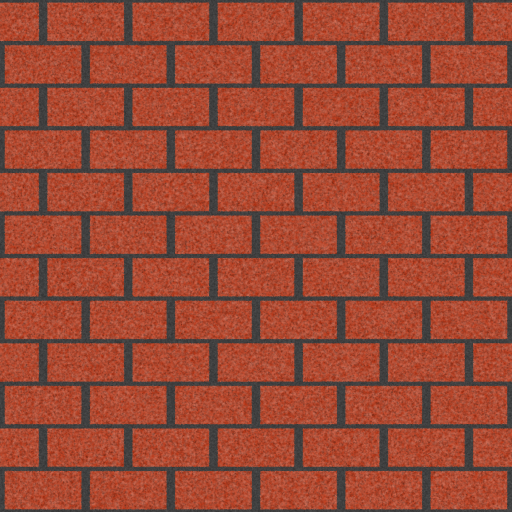
Згідно з вказаним завданням, була розроблена програма для відображення поверхні Richmond's Minimal Surface у вигляді суцільних трикутників під час виконання практичного завдання №2.

Зображення, що містить знімок екрана, мистецтво

Автоматично згенерований опис

Вигляд поверхні

Я обрав наступне зображення розміром 512х512 пікселів у форматі .jpg для подальшої розрахунково-графічної роботи. Це обрано з метою забезпечення сумісності з більшою кількістю операційних систем та браузерів.



Зображення текстури

Була реалізована накладання текстури на поверхню шляхом підготовки буфера текстурних координат. Кожна текстурна координата відповідає елементу масиву вершинного буфера. Згідно з варіантом, текстура повинна обертатися, тому було створено відповідний uniform для визначення кута обертання. У програмі шейдерів для визначення кольору пікселя фігури, на яку накладено текстуру, використовується функція texture2D(). Ця функція приймає об'єкт класу sampler2D як перший аргумент, що містить дані про зображення, та текстурну координату як другий аргумент.

Зображення, що містить мистецтво

Автоматично згенерований опис

Текстура накладена на поверхню

Для відображення точки, щодо якої буде виконуватися трансформація текстури, був створений новий об'єкт класу Model. Цей об'єкт графічно представлений у вигляді сфери та знаходиться на поверхні.

Зображення, що містить малюнок, мистецтво, творчість, ілюстрація

Автоматично згенерований опис

Точка на текстурованій поверхі

Згідно варіанту було імплементовано обертання текстури відносно точки на поверхні.

**Інструкції користувача**

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

Зображення, що містить мистецтво

Автоматично згенерований опис Зображення, що містить мистецтво, малюнок, дизайн, ілюстрація

Автоматично згенерований опис  
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.

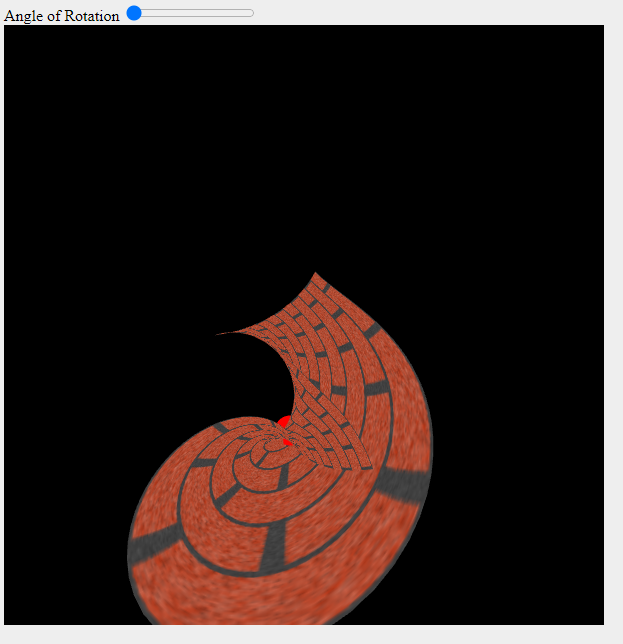
Зображення, що містить мистецтво

Автоматично згенерований опис Зображення, що містить мистецтво

Автоматично згенерований опис

Демонстрація переміщення точки

За допомогою слайдера з підписом “ Angle of Rotation” можна змінювати кут обертання відносно умовної точки на поверхні.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.

 Зображення, що містить знімок екрана, мистецтво

Автоматично згенерований опис

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 p = 2;

let projection = m4.orthographic(-p, p, -p, p, -p, p);

/\* 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, 0);

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. \*/

gl.uniform4fv(shProgram.iColor, [...hexToRgb(document.getElementById('cs').value), 1]);

gl.uniform3fv(shProgram.iColorLight, hexToRgb(document.getElementById('cl').value));

gl.uniform3fv(shProgram.iLightPosition, [2 \* Math.cos(Date.now() \* 0.001), 2 \* Math.sin(Date.now() \* 0.001), 0.1]);

gl.uniform2fv(shProgram.iPoint, point);

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

surface.Draw();

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

m4.translation(...vertex(mapRange(point[0], 0, 1, -1, 1), mapRange(point[1], 0, 1, 0.2, 1)))));

gl.uniform3fv(shProgram.iLightPosition, [0, 0, 0]);

light.Draw();

}

Код функції частини програми на GLSL

void main() {  
 mat4 m1 = translation(-point);  
 mat4 m2 = rotation(angle);  
 mat4 m3 = translation(point);  
 vec4 v1 = m1\*vec4(texture,0.,1.0);  
 vec4 v2 = m2\*v1;  
 vec4 v3 = m3\*v2;  
 newTexture = v3.st;  
 vec3 N = normalize(normal);  
 vec3 L = normalize(lightPosition - 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(-vertex); // Vector to viewer  
 // Compute the specular term  
 float specAngle = max(dot(R, V), 0.0);  
 specular = pow(specAngle, 50.0);  
 }  
 newColor = color.rgb\*0.1+lambertian\*color.rgb+lightColor\*specular;  
 gl\_Position = ModelViewProjectionMatrix \* vec4(vertex,1.0);  
}`;