**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-31мп

Пархомчук Дмитро Юрійович

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 відтворення інтерактивної 2D та 3D графіки у веб-браузерах. Воно надає функціонал для відображення текстур, масштабування, обертання та інтерактивного переміщення точок у веб-середовищі. WebGL дозволяє створювати візуально ефективні 2D та 3D графічні застосунки в браузері та взаємодіяти з іншими технологіями, такими як HTML та CSS, для створення повноцінних веб-застосунків. Для роботи з графікою використовуються функції WebGL, які дозволяють відображати текстури, масштабувати об'єкти, обертати їх та взаємодіяти з графічними елементами. WebGL вимагає підтримки веб-браузером і зазвичай підтримується більшістю сучасних браузерів. Розробка графічних застосунків на WebGL здійснюється за допомогою мови програмування JavaScript.

Шейдери WebGL

Шейдери в WebGL грають важливу роль у візуалізації 3D-графіки, забезпечуючи гнучкий та програмований підхід до визначення вигляду графічних об'єктів. Вершинні шейдери обробляють кожну вершину 3D-моделі, використовуючи атрибути, такі як положення, колір та координати текстури, для створення виходу. Вони відповідають за трансформації, такі як переміщення, обертання та масштабування.

Фрагментні шейдери працюють з кожним пікселем на екрані, отримуючи інтерпольовані значення від вершинного шейдера. Вони визначають кінцевий колір кожного пікселя, враховуючи освітлення та інші ефекти. Шейдери пишуться мовою GLSL та підлягають компіляції та об'єднанню в програму шейдера перед використанням. Uniforms представляють постійні значення для всіх вершин чи фрагментів, передаючи дані, такі як матриці трансформації чи освітлення.

Attributes це дані, що відрізняються між вершинами, і використовуються для передачі інформації, такої як положення вершин, нормалі та координати текстури. Всі ці аспекти шейдерів в WebGL дозволяють точно контролювати відображення графічних об'єктів, забезпечуючи високий рівень керованості та гнучкість у визначенні вигляду та взаємодії з графічними елементами.

Начало формы

Накладання текстур. Текстурні координати

Накладання текстур у WebGL є технікою в комп'ютерній графіці, що дозволяє реалістично відтворювати поверхні, додавши зображення або текстури до 3D-моделей. Цей процес включає в себе використання текстурних координат для точного позиціонування текстури на поверхні об'єкта.

Текстурні координати, зазвичай позначені як (u, v), є параметрами, які асоціюються з кожною вершиною 3D-об'єкта. Вони варіюються від 0 до 1 і використовуються для звернення до конкретних точок на зображенні текстури. У контексті WebGL, координати текстури пов'язані з кожною вершиною 3D-моделі та інтерполюються по всій поверхні під час візуалізації. Це гарантує точне охоплення текстури моделлю і створює реалістичний вигляд.

Під час використання у фрагментному шейдері, координати текстури можуть використовуватися для вибірки кольорів із текстури. Це дозволяє створювати деталізовані та реалістичні візуальні ефекти, додаючи текстурні зображення до поверхонь об'єктів. Текстурні координати грають важливу роль у покращенні візуальної якості та реалізму графічних сцен у веб-браузерах за допомогою WebGL.

Начало формы

**Розробка програми**

За варіантом моя фігура це “Astroidal Torus”. Під час виконання другого практичного завдання я розробив програму, яка відображає поверхню у формі цілісних трикутників.



Рисунок 1 - Вигляд поверхні

Я вибрав наступне зображення для РГР. Для забезпечення сумісності з більшою кількістю операційних систем та браузерів було обрано зображення розміром 512x512 пікселів у форматі .jpg.

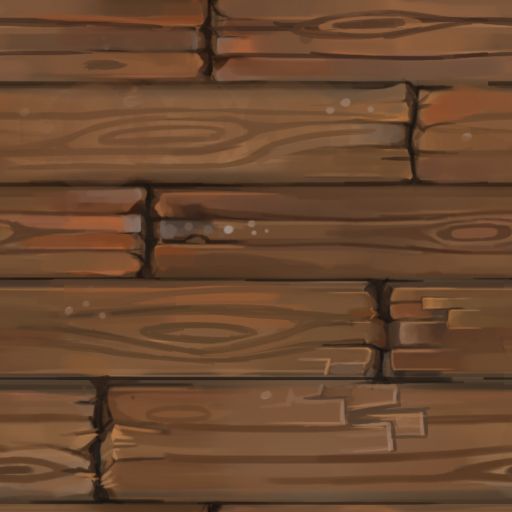
****

Рисунок 2 - Текстура

Була накладена текстура. Для накладання текстури був підготований буфер текстурних координат, де кожна з них відповідала елементу масиву з буфера вершин. Згідно варіанту, текстура повинна масштабуватися, тому був створений відповідний параметр "uniform", який визначає фактор масштабування. У програмі шейдера для визначення кольору пікселя фігури, на яку застосовано текстуру, необхідно використовувати функцію texture2D(). Ця функція приймає об'єкт класу sampler2D як перший аргумент, який зберігає дані про зображення, та текстурну координату як другий аргумент.

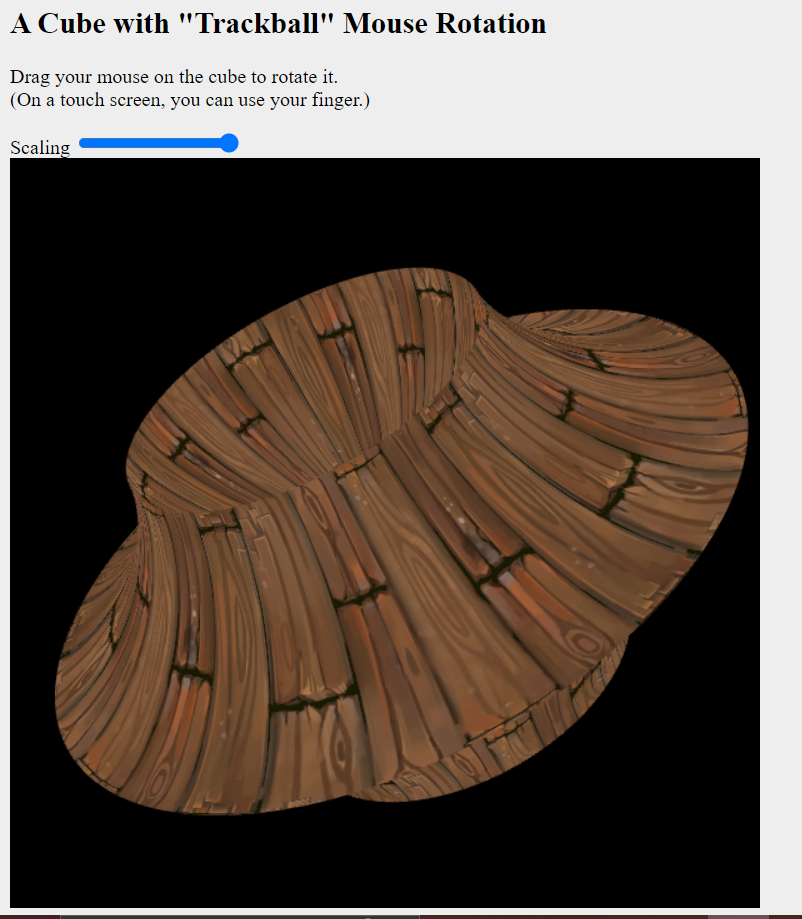


Рисунок 3 – Текстура, яку накладено на поверхню

Створено новий екземпляр класу Grap\_Model для відображення точки, відносно якої буде здійснюватися трансформація текстури. Графічне відображення об'єкта представлено у вигляді сфери, яка розташована на поверхні.



Рисунок 4 - Біла точка на текстурованій поверхні

Згідно завдання, було реалізовано масштабування текстури відносно точки на поверхні.

**Інструкція**

Для того щоб обертати фігуру, достатньо затиснути ліву кнопку миші та перетягнути її у напрямку, в якому бажаєте виконати обертання, відносно центру.

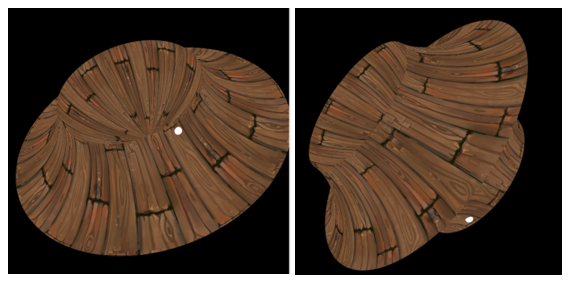


Рисунок 5 – Процес обертання

Для здійснення переміщення точки, відносно якої відбувається обертання, використовуйте клавіші WASD. Кожне натискання клавіші призводить до зсуву точки по поверхні на певну відстань. Переміщення відбувається до досягнення встановленого обмеження.

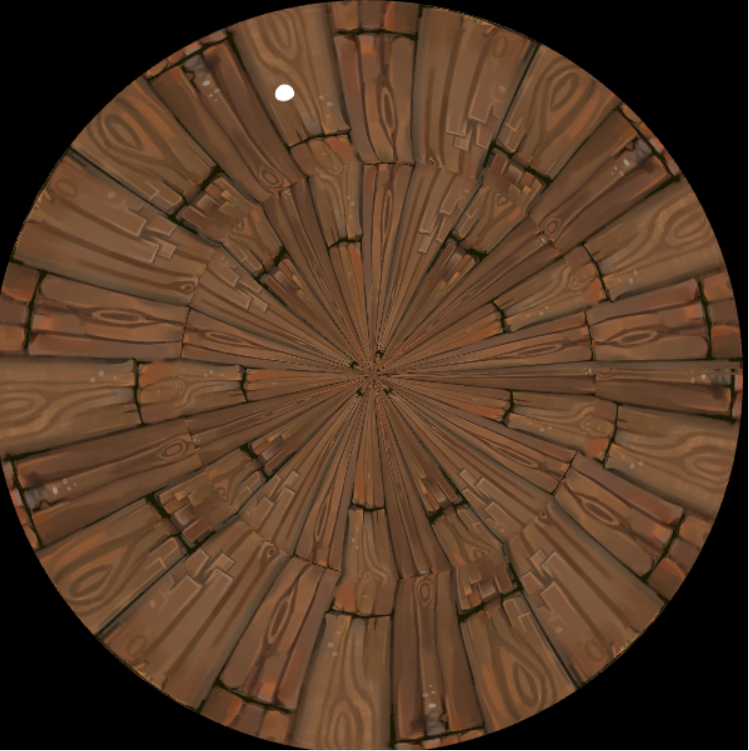
 

Рисунок 6 - Демонстрація переміщення точки відносно нерухомої фігури

За допомогою слайдера з назвою "Scaling" можна регулювати фактор масштабування відносно визначеної точки на поверхні. Важливо відзначити, що при переміщенні точки, текстура також зсувається, оскільки масштабування відбувається відносно іншої точки на поверхні, яка, в свою чергу, відповідає іншій текстурній координаті.

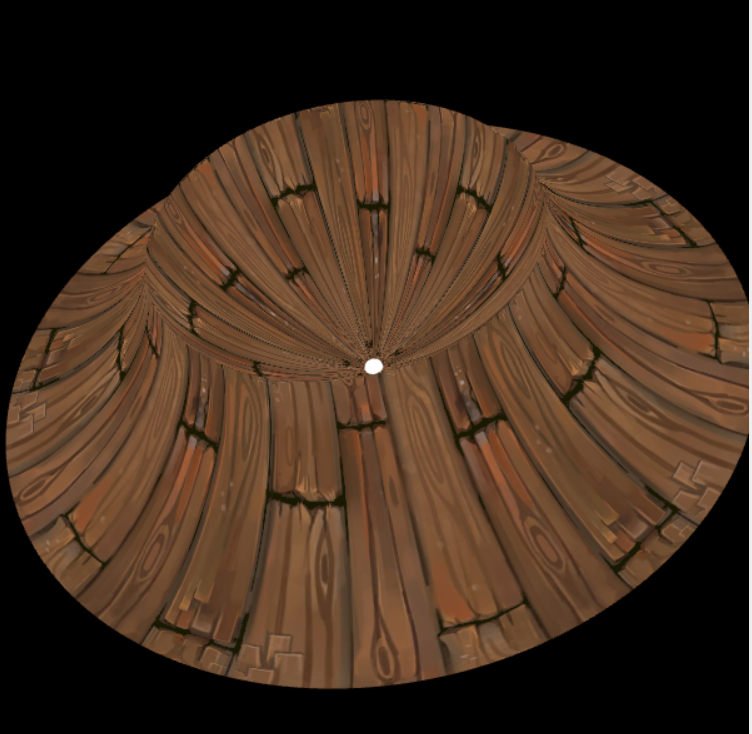
 

Рисунок 7 – Зміна фактору масштабування текстури

При перезавантаженні сторінки значення обертання фігури і фактору масштабування повернуться до стандартних. Положення точки відносно поверхні також буде скинуто до значення за замовчуванням.

Начало формы

**Код**

Код функції частини програми

function draw() {

    gl.clearColor(0, 0, 0, 1);

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

    const getVert = (u, v) => {

        const a = 1;

        const r = 1;

        const theta = 0;

        const x = (r + a \* Math.pow(Math.cos(u), 3) \* Math.cos(theta) - a \* Math.pow(Math.sin(u), 3) \* Math.sin(theta)) \* Math.cos(v);

        const y = (r + a \* Math.pow(Math.cos(u), 3) \* Math.cos(theta) - a \* Math.pow(Math.sin(u), 3) \* Math.sin(theta)) \* Math.sin(v);

        const z = a \* Math.pow(Math.cos(u), 3) \* Math.sin(theta) + a \* Math.pow(Math.sin(u), 3) \* Math.cos(theta);

        return [x, y, z];

    };

    gl.clearColor(0, 0, 0, 1);

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

    const projection = m4.perspective(Math.PI / 8, 1, 8, 20);

    const modelView = spaceball.getViewMatrix();

    const rotateToPointZero = m4.axisRotation([0.707, 0.707, 0], 0.0);

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

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

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

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

    const normal = m4.identity();

    m4.inverse(modelView, normal);

    m4.transpose(normal, normal);

    gl.uniformMatrix4fv(sh\_prog.iNormalMatrix, false, normal);

    gl.uniformMatrix4fv(sh\_prog.iModelViewProjectionMatrix, false, modelViewProjection);

    gl.uniform4fv(sh\_prog.iColor, [1, 1, 0, 1]);

    const dx = document.getElementById('dx').value;

    const dy = document.getElementById('dy').value;

    const dz = document.getElementById('dz').value;

    const px = document.getElementById('px').value;

    const pz = document.getElementById('pz').value;

    gl.uniform3fv(sh\_prog.iLDir, [dx, dy, dz]);

    gl.uniform3fv(sh\_prog.iLPos, [px, 3 \* Math.sin(Date.now() \* 0.001), pz]);

    gl.uniform1f(sh\_prog.iLimit, parseFloat(document.getElementById('lim').value));

    gl.uniform1f(sh\_prog.iEasing, parseFloat(document.getElementById('ease').value));

    gl.uniform2fv(sh\_prog.iTT, parameters);

    geometry.Draw();

    gl.uniform1f(sh\_prog.iLimit, -100.0);

    const newModelViewProjection = m4.multiply(modelViewProjection, m4.translation(...getVert((parameters[0] - 0.5) \* Math.PI \* 2, parameters[1] \* Math.PI \* 2)));

    gl.uniformMatrix4fv(sh\_prog.iModelViewProjectionMatrix, false, newModelViewProjection);

    lightSource1.Draw();

}