**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 - 24

Виконав: студент групи ТР-31мп

Підлісний Ярослав Андрійович

Перевірив: Демчишин А. А.

Київ - 2023

**Завдання**

1. Нанести текстуру на поверхню з практичного завдання №2.
2. Реалізувати масштабування/обертання текстури (координати текстури) масштабування/обертання навколо визначеної користувачем точки – непарні варіанти реалізують масштабування, парні варіанти реалізують обертання.
3. 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

**Теоритична складова**

*WebGL*

WebGL (Web Graphics Library) є веб-стандартом, який дозволяє відображати 3D-графіку у веб-браузерах без використання плагінів.

Основні характеристики WebGL:

1.WebGL використовує шейдери, написані на мові GLSL, і надає багатий набір функцій для рендерингу інтерактивної 3D-графіки.

2.WebGL тісно інтегрований з іншими веб-технологіями, дозволяючи використовувати HTML, CSS і JavaScript для створення інтерактивних 3D-додатків.

3.Більшість сучасних веб-браузерів підтримують WebGL, що робить його доступним для широкого кола користувачів.

4.Розробка для WebGL здійснюється з використанням JavaScript, що робить його доступним для широкої аудиторії веб-розробників.

5.Оскільки WebGL працює в браузері, додатки на його основі легко доступні на різних платформах, включаючи мобільні пристрої.

WebGL часто використовується для створення ігор, візуалізацій даних, архітектурних візуалізацій та інших інтерактивних 3D-додатків у веб-середовищі.

*WebGL Shaders*

WebGL використовує шейдери для створення деталізованої 3D-графіки, причому два основні типи шейдерів у WebGL - це вершинні та фрагментні шейдери.

Вершинні шейдери займаються обробкою кожної вершини у 3D-моделі. Вони приймають атрибути, такі як положення, колір і текстурні координати вершин, і перетворюють їх. Ці шейдери виконують такі операції, як переміщення, обертання та масштабування вершин, що дозволяє правильно розмістити об’єкти в 3D-просторі. Вихідні дані цих шейдерів передаються до фрагментного шейдера і включають перетворену позицію вершини та інші обчислені значення.

Фрагментні шейдери, відомі також як піксельні шейдери, обробляють кожен піксель, що має бути намальований на екрані. Вони використовують дані, отримані від вершинних шейдерів, включаючи координати кольору та текстури, а також інформацію про освітлення, для визначення кінцевого кольору пікселя. Цей процес може включати вибірку з текстур та обчислення освітлення для створення реалістичних ефектів.

Обидва типи шейдерів написані на GLSL (OpenGL Shading Language) і мають бути скомпільовані перед використанням. Вони об’єднуються в шейдерну програму, яка повинна бути зв’язана з конвеєром відтворення WebGL.

Щодо інтерфейсу з програмою, то використовуються uniforms та attributes. Uniforms - це сталі значення протягом візуалізації, які можуть передавати шейдерам зовнішні дані, наприклад, матриці трансформації. Attributes - це змінні дані для кожної вершини, такі як її положення, нормалі, координати текстури, які використовуються вершинним шейдером. У фрагментних шейдерах координати текстури використовуються для вибірки кольорів з текстур, що впливає на кінцевий колір пікселя на екрані.

*Текстури*

Техніка відображення текстур у комп'ютерній графіці дозволяє досягти реалістичного зовнішнього вигляду об'єктів шляхом накладання текстурних зображень на 3D-моделі. У сфері WebGL, ця методика включає прив'язку унікальних текстурних координат (u, v) до кожної вершини 3D-об'єкта. Ці координати, варіюючись від 0 до 1, використовуються для точного визначення місця зображення текстури, яке має бути нанесене на модель. Такий підхід забезпечує деталізоване та точне відтворення текстур на поверхнях об'єктів.

Текстурні координати (u, v) є ключовими у визначенні того, яким чином текстура буде розташована на об'єкті. У WebGL, ці координати асоціюються з кожною вершиною моделі і рівномірно розподіляються по всій поверхні об'єкта під час його рендерингу. Такий процес забезпечує точне покриття моделі текстурою, додаючи деталей та глибини зображенню.

**Розробка**

За варіантом мені було надано Surface of Revolution

"Wellenkugel". При виконанні практичного завдання №2 було розроблено програму, що виводить поверхню у вигляді суцільних трикутників.

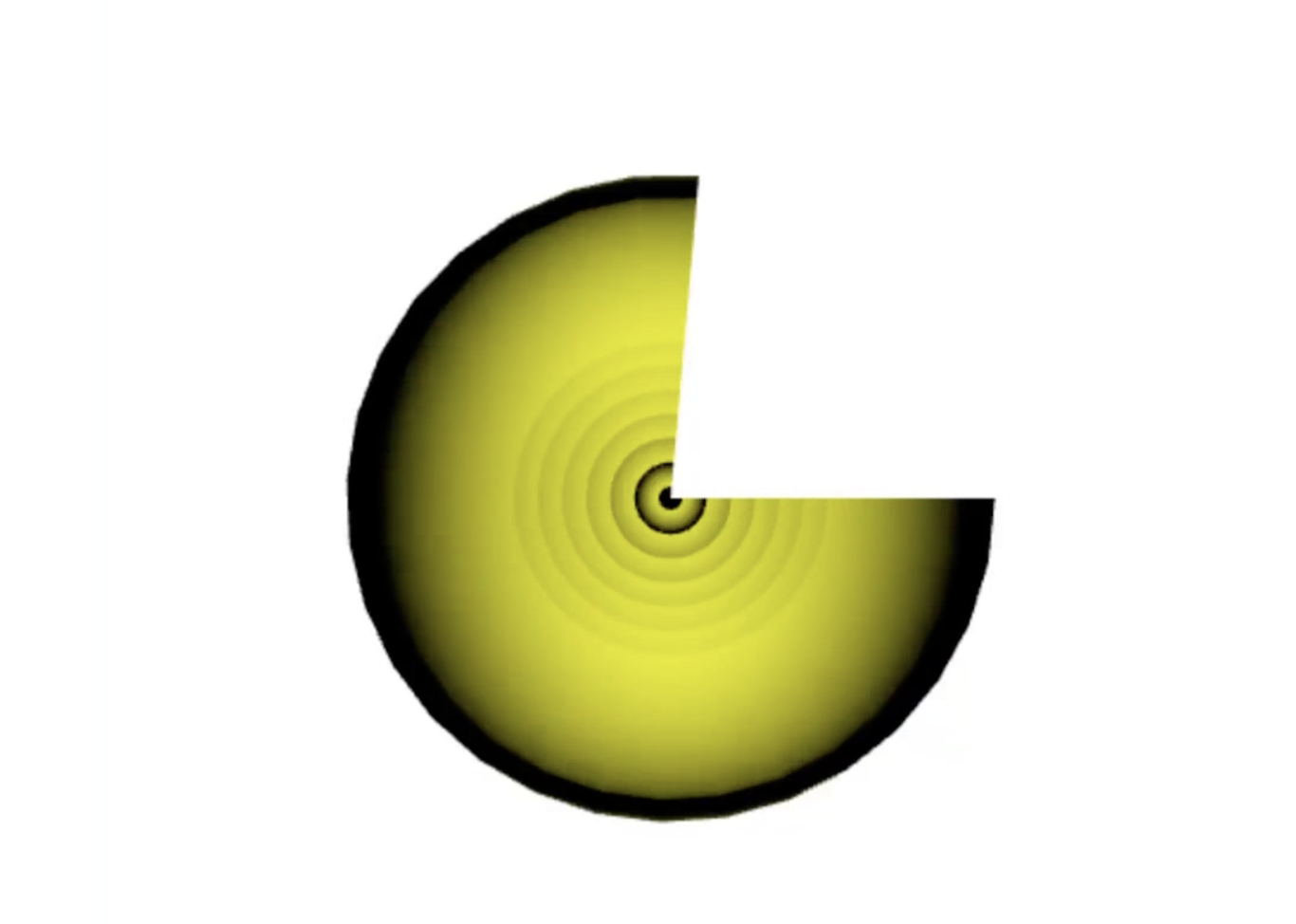


Рис. 1 - Поверхня

Було обрано наступне зображення для відображення текстури



Рис. 2 - Текстура

Була застосована текстура до поверхні, для чого було створено буфер текстурних координат, кожна з яких корелює з елементом у масиві вершин. Враховуючи, що текстура має обертатись (згідно варіанту), було додано спеціальний uniform для визначення кута обертання. У шейдерній програмі для визначення кольору пікселя на текстурованій поверхні використовується функція texture2D(). Ця функція бере два аргументи: об'єкт класу sampler2D, який містить інформацію про зображення, та текстурну координату, яка визначає точку на зображенні, з якої потрібно взяти колір.

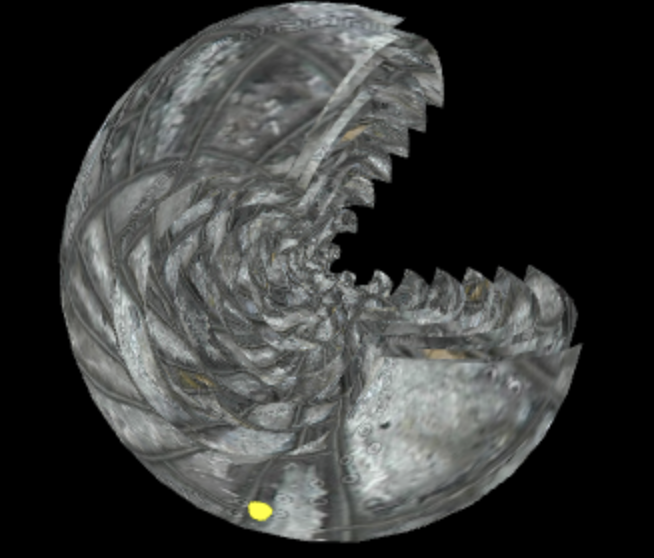


Рис. 3 - Текстура на поверхні

Було створено новий обʼєкт класу Model для відображення точки відносно якої буде здійснюватись трансформація текстури. Оюєкт являє собою шар на поверхні фігури



Рис. 4 - Точка на текстурованій поверхні

Згідно варіанту було імплементовано обертання текстури відносно точки на поверхні.

**Інструкції для користувача**

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

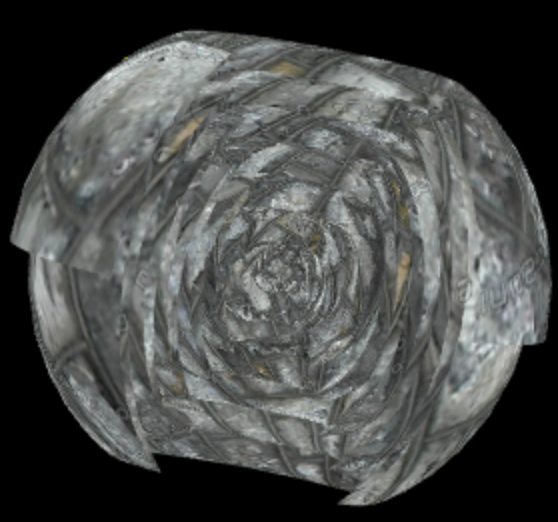
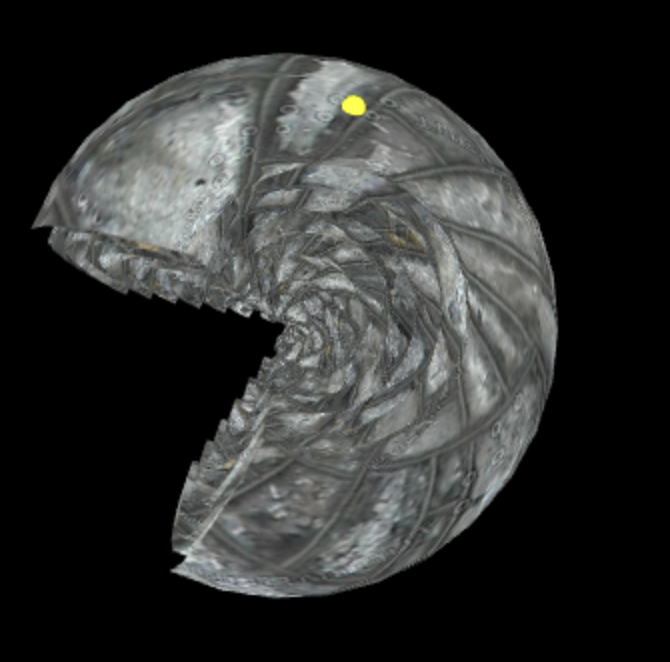


Рис. 5 - Вигляд фігури до та після обертання

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



Рис. 6 - Демонстрація переміщення точки

Використовуючи слайдер із назвою "Speed of Rotation", можна регулювати кут обертання навколо уявної точки на поверхні. При встановленні кута обертання, відмінного від нуля, стає помітним, що переміщення цієї точки спричиняє зсув текстури. Це відбувається через те, що обертання проводиться відносно іншої точки на поверхні, яка кореспондує з різною текстурною координатою.



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.

**Лістинг 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.orthographic(-5, 5, -5, 5, -5, 5);

/\* 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, -3);

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, [1, 1, 0, 1]);

surface.Draw();

let lim\_u = 14.5 \* Math.PI \* 100;

let lim\_v = 1.5 \* Math.PI \* 100;

gl.uniform2fv(shProgram.iPointUV, [map(pointUV.u, 0, lim\_u, 0, 10), map(pointUV.v, 0, lim\_v, 0, 1)])

let x = (pointUV.u / 100) \* Math.cos(Math.cos(pointUV.u / 100)) \* Math.cos(pointUV.v / 100);

let y = (pointUV.u / 100) \* Math.cos(Math.cos(pointUV.u / 100)) \* Math.sin(pointUV.v / 100);

let z = (pointUV.u / 100) \* Math.sin(Math.cos(pointUV.u / 100));

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

modelViewProjection,

m4.translation(x / 20, y / 20, z / 20)));

gl.uniform1i(shProgram.iPoint, true)

gl.uniform1f(shProgram.iAngle, angle)

pointModel.Draw()

gl.uniform1i(shProgram.iPoint, false)

}

void main() {

vec4 texColor = texture2D(tmu,v\_texture);

gl\_FragColor = texColor;

if(point){

gl\_FragColor = color;

}

}`;