МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО» ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ КАФЕДРА ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Графічна робота з дисципліни « *Візуалізація графічної та геометричної інформації*»

Виконала: студентка 5-го курсу ІАТЕ
групи ТР-21мп

Краєвська Марія Дмитрівна

Перевірив:

Демчишин Анатолій Анатолійович

Київ-2022

1. Опис завдання

Завданням графічної роботи є відображення текстури на поверхні фігури, відтвореної в попередніх лабораторних відповідно до варіанту. До даної роботи були виставлені наступні вимоги:

1. Текстура має бути нанесена на поверхню, яка виконана в другому практичному завданні.
2. Виконана текстура має обертатися навколо заданої точки.
3. За допомогою елементів керування, а саме клавіш W, S , A, D , необхідно реалізувати переміщення точки по поверхні.
4. Додати анімацію виконаної роботи в проект.
5. Надати звіт до роботи та викласти результат в свій репозиторій.

2. Теоритичні відомості

Текстура в розумінні проектування поверхні в WebGL – це спосіб надання поверні фізичних властивостей. В нашому випадку текстурою буде виступати картинка завантажена з мережі Інтернет. За накладання текстури відповідає модуль відображення текстури(TMU).

Текстурний блок, який також називають блоком відображення текстури (TMU) або блоком обробки текстури (TPU), є апаратним компонентом в графічному процесорі, який виконує вибірку(sampling).

Вибірка(sampling) - це процес обчислення кольору з текстури зображення та координат текстури. Відображення зображення текстури на поверхню є досить складною операцією, оскільки вимагає більше, ніж просто повернення кольору текселя, який містить деякі задані координати текстури.

Семплер (наприклад, sampler2D) є фактично непрозорими дескриптором текстур. Він використовується з вбудованими функціями текстур,щоб вказати, до якої текстури потрібно отримати доступ. Вони можуть бути оголошені тільки як параметри функції або форми.

Функції пошуку текстур доступні як для вершинних, так і для фрагментних шейдерів. Однак, рівень деталізації не обчислюється фіксованою функціональністю для вершинних шейдерів, тому існують деякі відмінності у роботі між вершинним та фрагментним пошуком текстур.

Функції, наведені у таблиці 1 нижче, надають доступ до текстур через семплери, які налаштовуються через OpenGL ES API. Властивості текстури, такі як розмір, формат пікселів, кількість розмірність, метод фільтрації, кількість рівнів міп-карти, порівняння глибини тощо, також визначаються викликами викликами OpenGL ES API.

Syntax	Description
vec4 **texture2D** (sampler2D *sampler*, vec2 *coord*) vec4 **texture2D** (sampler2D *sampler*, vec2 *coord*, float *bias*) vec4 **texture2DProj** (sampler2D *sampler*, vec3 *coord*) vec4 **texture2DProj** (sampler2D *sampler*, vec3 *coord*, float *bias*) vec4 **texture2DProj** (sampler2D *sampler*, vec4 *coord*) vec4 **texture2DProj** (sampler2D *sampler*, vec4 *coord*, float *bias*) vec4 **texture2DLod** (sampler2D *sampler*, vec2 *coord*, float *lod*) vec4 **texture2DProjLod** (sampler2D *sampler*, vec3 *coord*, float *lod*) vec4 **texture2DProjLod** (sampler2D *sampler*, vec4 *coord*, float *lod*)	Use the texture coordinate *coord* to do a texture lookup in the 2D texture currently bound to *sampler*. For the projective ("**Proj**") versions, the texture coordinate (*coord.s, coord.t*) is divided by the last component of *coord*. The third component of coord is ignored for the vec4 coord variant.

Таблиця 1. Функції пошуку текстур

Функції, що містять параметр bias, доступні тільки у фрагментному шейдері. Якщо зсув присутній, то він додається до розрахованого рівня деталізації перед виконанням операції доступу до текстури.

Якщо параметр bias не передбачено, то реалізація вибирає рівень деталізації автоматично.

Вбудовані функції з суфіксом "Lod" дозволені тільки у вершинному шейдері. Для функцій "Lod", lod безпосередньо використовується як рівень деталізації.

Накладання 2D текстури на 3D фігуру відбувається за допомогою UV мапінгу. На кожну вершину просторової поверхні, накладається координати текстури(UV координати).

3. Виконання практичної частини

В першу чергу для виконання графічної роботи, необхідно підготувати зображення. Було знайдено ресурси з відкритим доступом до зображень та вибрано необхідну картинку, а саме текстуру лави(Рис. 1).

Коли ми рендеримо модель, ми хочемо, щоб дані моделі зберігалися в пам'яті графічного процесора, щоб вони були безпосередньо доступні для програми шейдерів. Щоб використовувати зображення як таблицю для пошуку значень кольорів, нам потрібно, щоб зображення також було доступне з пам'яті графічного процесора.

Виконаємо наступні кроки для створення об'єкта текстури:

- Створюємо новий об'єкт текстури.
- Встановлюєм параметри, які контролюють використання об'єкта текстури.
- Копіюємо зображення в об'єкт текстури.

```
const image = new Image();
image.crossOrigin = 'anonymous';
image.src = 'https://www.the3rdsequence.com/texturedb/download/259/texture/jpg/256/burning+hot+lava-256x256.jpg';
image.onload = () => {
  const texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
```

Рисунок 1. Завантаження та створення текстури

Перейдемо до шейдерів, які виконують відображення текстури. Вершинний шейдер копіює координати текстури вершини у змінну, щоб їх можна було інтерполювати по поверхні трикутника. Координати текстури обчислюються в функції, відповідно до заданого кута повороту та вибранної точки обертання.

```
vec2 getTexture(vec2 texture, float Angle, vec2 Point) {
    mat4 rotateMat = getRotate(Angle);
    mat4 translate = getTranslate(-Point);
    mat4 translateBack = getTranslate(Point);

    vec4 textCoordTr = translate * vec4(texture, 0, 0);
    vec4 textCoordRotate = textCoordTr * rotateMat;
    vec4 textCoordTrBack = textCoordRotate * translateBack;

    return vec2(textCoordTrBack);
}
```

Рисунок 2. Отримання координат текстури

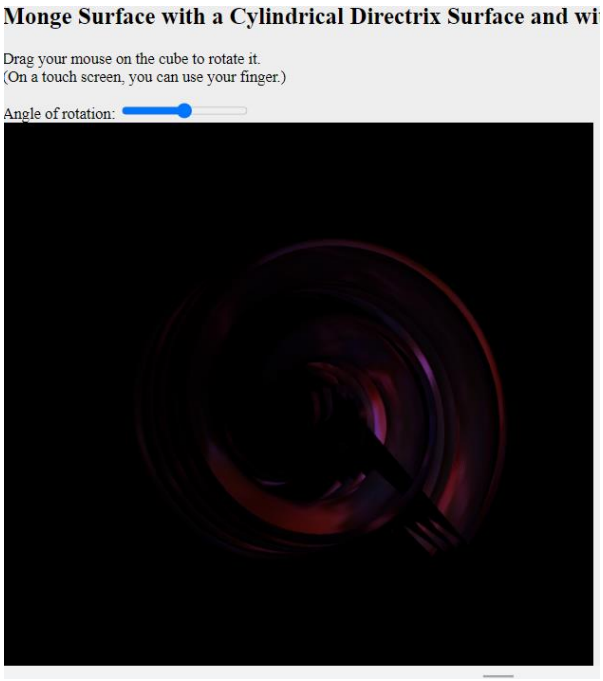
Шейдер фрагмента використовує координати текстури для фрагмента, щоб знайти колір на зображенні карти текстури. Це звичайна операція, яка вбудована у функціонал GLSL. Тож викликаємо функцію texture2D і вказуєте текстурну одиницю для використання та координати текстури.

```
vec4 texture = texture2D(uTexture, v_texture);
```

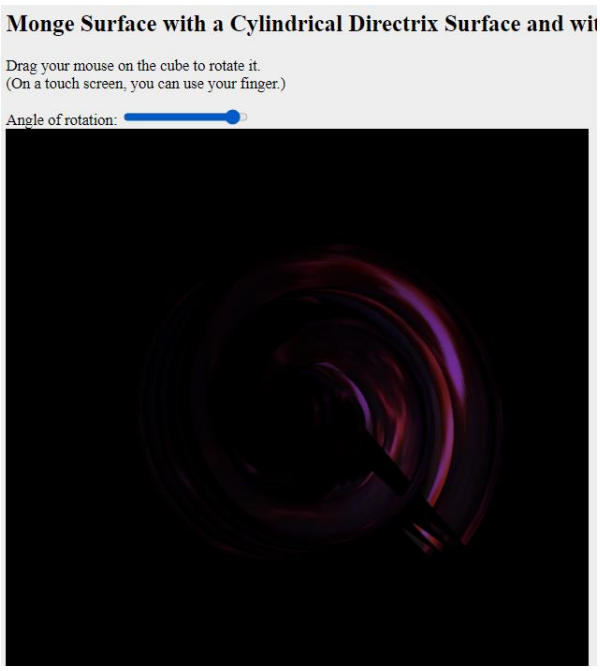
4. Інструкція використання

Виконана графічна робота має декілька важелів управління. Перший – це кут обертання. За допомогою миші користувач може перетягнути повзунок та змінити кут. Відобразимо роботу обертання на наступних зображеннях:

До



Після



Також для користувача доступна зміна координати точки на поверхні, відносно якої можна маштабувати та обертати текстуру. Управління координатами U та V виконується за допомогою клавіш W та S , A та D. Натискання на ці клавіші збільшує та зменшує значення координати відповідно.

5. Лістинг коду

```
function CreateSurfaceData() {
    let vertexList = [];
    let normalsList = [];
    let textureList = [];

    let deltaT = 0.0005;
    let deltaA = 0.0005;

    const step = 0.1

    for (let t = -15; t <= 15; t += step) {
        for (let a = 0; a <= 15; a += step) {
            const tNext = t + step;
            vertexList.push(getX(t, a, 10), getY(t, a,
10), getZ(t, 20));
            vertexList.push(getX(tNext, a, 10),
getY(tNext, a, 10), getZ(tNext, 20));

            let result = m4.cross(calcDerT(t, a,
deltaT), calcDerA(t, a, deltaA));
            normalsList.push(result[0], result[1],
result[2])

            result = m4.cross(calcDerT(tNext, a,
deltaT), calcDerA(tNext, a, deltaA));
            normalsList.push(result[0], result[1],
result[2]);


            textureList.push(...[(t + 15) / 30, a /
15]);
            textureList.push(...[(tNext + 15) / 30,(
a+step) / 15]);
        }
    }


    return {vertexList, normalsList, textureList};
}
```

```
// Vertex shader
const vertexShaderSource = `
attribute vec3 vertex;
attribute vec3 normal;
attribute vec2 texture;

uniform mat4 ModelViewProjectionMatrix;
uniform mat4 WorldInverseTranspose;
uniform mat4 WorldMatrix;
uniform vec3 LightWorldPosition;
uniform vec3 ViewWorldPosition;
uniform vec2 Point;
uniform float Angle;
varying vec3 v_normal;
varying vec3 v_surfaceToLight;
varying vec3 v_surfaceToView;
varying vec2 v_texture;


mat4 getRotate(float angleToRad) {
  float c = cos(angleToRad);
```

```
  float s = sin(angleToRad);


  return mat4(
    vec4(c, s, 0.0, 0.0),
    vec4(-s, c, 0.0, 0.0),
    vec4(0.0, 0.0, 1.0, 0.0),
    vec4(0.0, 0.0, 0.0, 1.0)
  );
}


mat4 getTranslate(vec2 t) {
  return mat4(
    1.0, 0.0, 0.0, t.x,
    0.0, 1.0, 0.0, t.y,
    0.0, 0.0, 1.0, 0.0,
    0.0, 0.0, 0.0, 1.0
  );
}


vec2 getTexture(vec2 texture, float Angle, vec2
Point) {
    mat4 rotateMat = getRotate(Angle);
    mat4 translate = getTranslate(-Point);
    mat4 translateBack = getTranslate(Point);
```

```
    vec4 textCoordTr = translate * vec4(texture, 0,
0);

    vec4 textCoordRotate = textCoordTr * rotateMat;

    vec4 textCoordTrBack = textCoordRotate *
translateBack;



    return vec2(textCoordTrBack);

}



void main() {

    gl_Position = ModelViewProjectionMatrix *
vec4(vertex,1.0);

    v_normal = mat3(WorldInverseTranspose) * normal;

    vec3 surfaceWorldPosition = (WorldMatrix *
vec4(vertex, 1.0)).xyz;

    v_surfaceToLight = LightWorldPosition -
surfaceWorldPosition;

    v_surfaceToView = ViewWorldPosition -
surfaceWorldPosition;

    v_texture = getTexture(texture, Angle, Point);

}`;




// Fragment shader

const fragmentShaderSource = `

#ifdef GL_FRAGMENT_PRECISION_HIGH

   precision highp float;
```

```
#else

   precision mediump float;

#endif

uniform vec4 color;

uniform vec3 LightDirection;

uniform float limit;

uniform sampler2D uTexture;

varying vec3 v_normal;

varying vec3 v_surfaceToLight;

varying vec3 v_surfaceToView;

varying vec2 v_texture;

void main() {

    vec3 normal = normalize(v_normal);

    vec3 surfaceToLightDirection =
normalize(v_surfaceToLight);

    vec3 surfaceToViewDirection =
normalize(v_surfaceToView);

    vec3 halfVector =
normalize(surfaceToLightDirection +
surfaceToViewDirection);



    float shininess = 8.0;



    float light = 0.0;

    float specular = 0.0;

    float dotFromDirection =
dot(surfaceToLightDirection, -LightDirection);
```

```
    if (dotFromDirection >= limit) {
      light = dot(normal, surfaceToLightDirection);
      if (light > 0.0) {
        specular = pow(dot(normal, halfVector),
shininess);
      }
    }
    vec4 texture = texture2D(uTexture, v_texture);
    gl_FragColor = texture * color;
    gl_FragColor.rgb *= light;
    gl_FragColor.rgb += specular;
}`;
```