Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Інститут атомної та теплової енергетики
Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота

З дисципліни «Візуалізація графічної та геометричної інформації»
Варіант 23

Виконала:
Притула Єлізавета
Студентка групи ТР-22мп

Київ-2022

1. Завдання

Тема роботи: Операціїї над тектурними координатами. Масштабування та обертання текстури навколо вказаної користувачем точки.

Вимоги:

- Накласти текступу на поверхню отриману в результаті виконання практичного завдання №2.
- Імплементувати масштабування текстури (координат текстури)/обертання навколо визначеної користувачем точки. непарні варіанти реалізують масштабування, парні варіанти - обертання.
- Запровадити можливість переміщення точки відносно якої відбувається трансформація текстури по поверхні за рахунок зміни параметрів в просторі текстури. Наприклад, клавіші A та D для переміщення по осі абсцис, змінюючи параметр u текстури, а клавіші W та S по осі ординат, змінюючи параметр v.

2. Теоретичні відомості

Текстура – це зображення (растровий формат), що застосовується до полігональної моделі шляхом накладання, з метою надання моделі фактурності, рельєфності та потрібного кольору.

Текстурування – важливий етап у процесі створення та візуалізації 3D-моделі виробу, що дозволяє надати поверхні об'ємного об'єкта певних параметрів та властивостей, для надання її максимальної реалістичності та подібності до реального об'єкта. Усі дрібніші візуальні характеристики у 3D-моделюванні, такі як зморшки та окремі нитки килима, є продуктом текстури, нанесеної художником.

Зазвичай створювані 3D-моделі мають стандартний сірий колір програми. Щоб додати кольори, малюнки та текстури, 2D-фотографії потрібно розмістити на 3D-моделях. Додавання кольорів або властивостей поверхні та матеріалу до 3D-моделі вимагає ще одного кроку вперед у процесі 3D-моделювання, тобто 3D-текстурування. Цей підхід часто призводить до повного кольору та властивостей поверхні 3D-моделі.

Щоб почати процес 3D-текстурування, необхідно спочатку розгорнути модель, що, по суті, те саме, що розгортання 3D-сітки. Коли художники-фактуристи отримають готові моделі від відділу 3D-моделювання, вони створять UV-карту для кожного 3D-об’єкта. UV-карта — це плоске зображення поверхні 3D-моделі, яке використовується для швидкого накладання текстур. Прямо пов’язуючи

2D-зображення (текстуру) з вершинами багатокутника, UV-відображення може допомогти обернути 2D-зображення (текстуру) навколо 3D-об’єкта, а згенеровану карту можна використовувати безпосередньо в процесі текстурування та затінення.

Більшість програмних систем 3D мають кілька інструментів або підходів для розгортання 3D-моделей. Коли справа доходить до створення UV-карт, це питання особистих уподобань. Якщо ви не збираєтеся використовувати процедурні текстури, майже завжди потрібно розгортати 3D-модель у компоненті текстурування. Це текстури, створені за допомогою математичних методів (процесів), а не безпосередньо записаних даних у 2D або 3D.

3. Виконання завдання

Під час виконання другого практичного завдання було створено поверню сполучення коаксіального циліндра та конуса. Отриману поверхню можна побачити на рисунку 3.1.

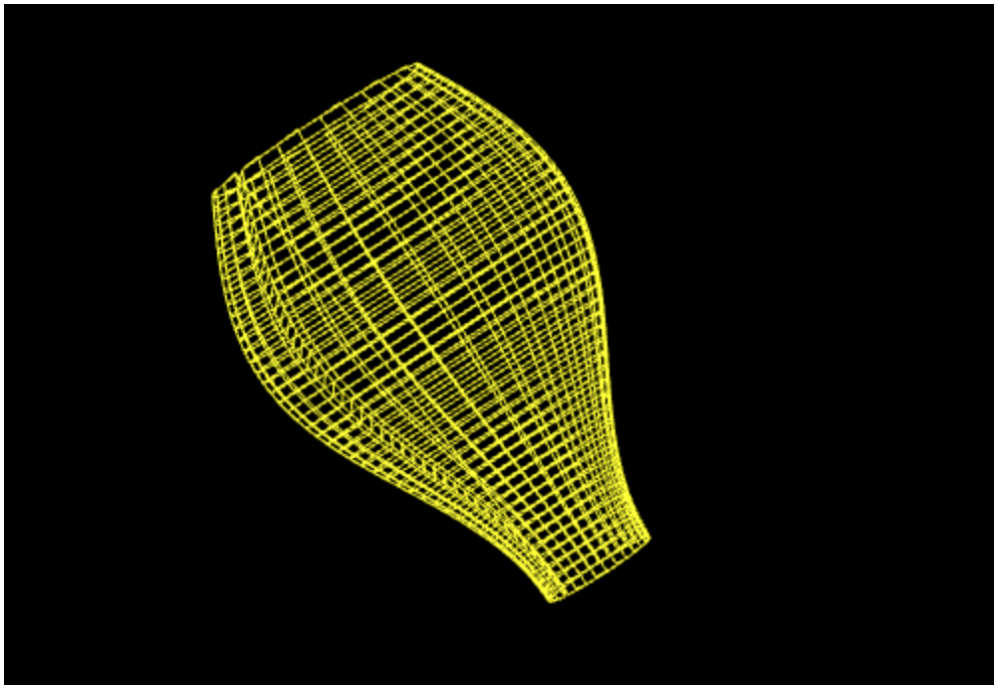

Рис. 3.1 поверхня сполучення коаксіального циліндра та конуса

Для відображення текстури було обрано картинку формату «jpg». Завантаживши її на github, щоб в подальшому легко використовувати посилання на неї.

В графічному редакторі було налаштувано розмір картинки так (512x512), щоб ширина і висота були рівні, а також, аби сторона мала розмір 2^n в пікселях.

З метою накладання текстури на поверхню, в першу чергу було створено декілька змінних в коді шейдера. Після чого були створення посилання на них в коді програми. Були також створені функції для генерації бUVера даних текстури.

Обрану картинку можна побачити на рисунку 3.2.

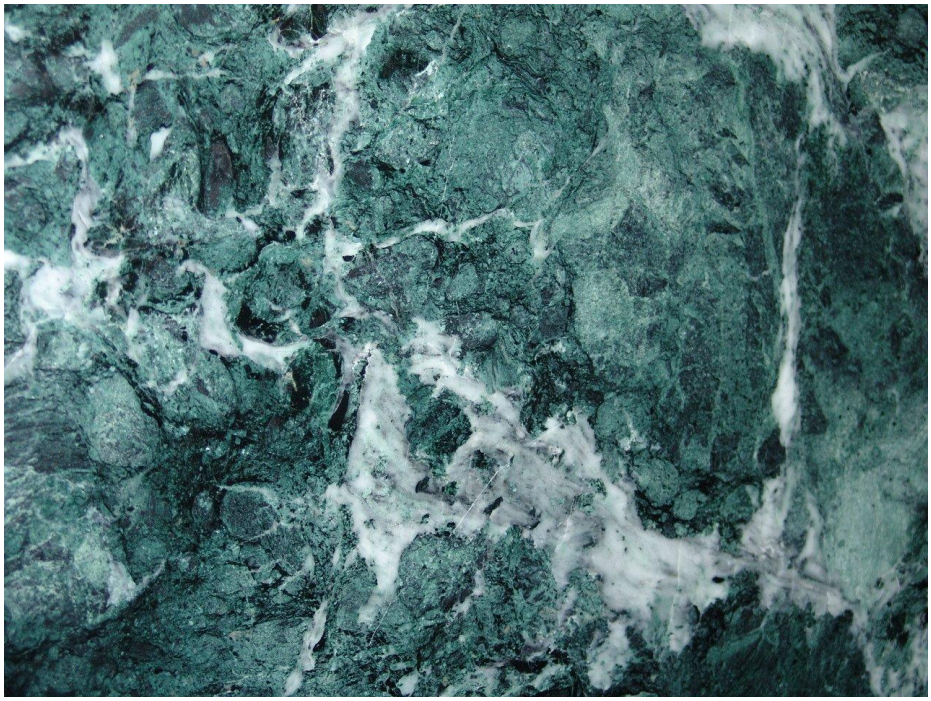

Рис. 3.2 Обрана текстура

Поверхню з накладеною текстурою можна побачити на рисунку 3.3

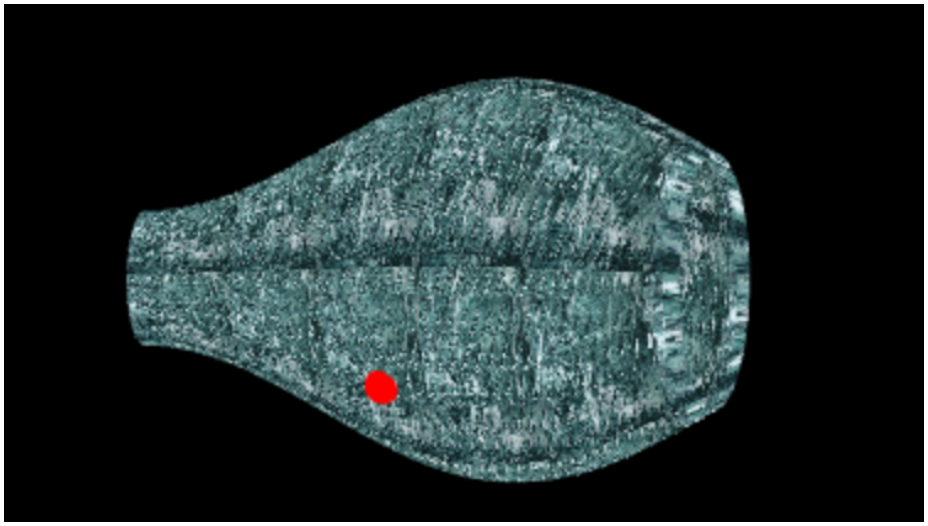

Рис. 3.3 поверхня сполучення коаксіального циліндра та конуса з накладеною текстурою

Для відображення умовної точки відносно якої буде виконватися трансформація текстури, в класі моделі було сворено відповідну функцію. Замість відображення точки було прийнято рішення відобжати сферу, адже працюємо в 3д-просторі. Для відображення сфери необхідно було створити функцію, яка б створювала геометрію для неї.

Для роботи з текстурою було створено ще кілька змінних в коді шейдера:

- обертання текстури, розташування умовної точки в (u,v) координатах,
- змінну для розташування сфери на відповідне місце поверхні в 3д-просторі.

Для реалізації переміщення точки по поверхні та обертання текстури було додано відповідні функції на відповідні вхідні дані від користувача.

4. Вказівки користувачу

Користувач може обертати поверхню відносно умовної точки, а також змінювати орієнтацією поверхні в просторі. Також можливе переміщення умовної точки по поверхні.

Переміщення умовної точки реалізовується за допомогою введення натискань клавіш W та S, які здійснюють переміщення точки за параметром v в додатньому та від'ємному напрамках відповідно, клавіші A та D здійснюють переміщення точки за параметром u у від'ємному та додатньому напрямках відповідно(рис 4.1-4.2).

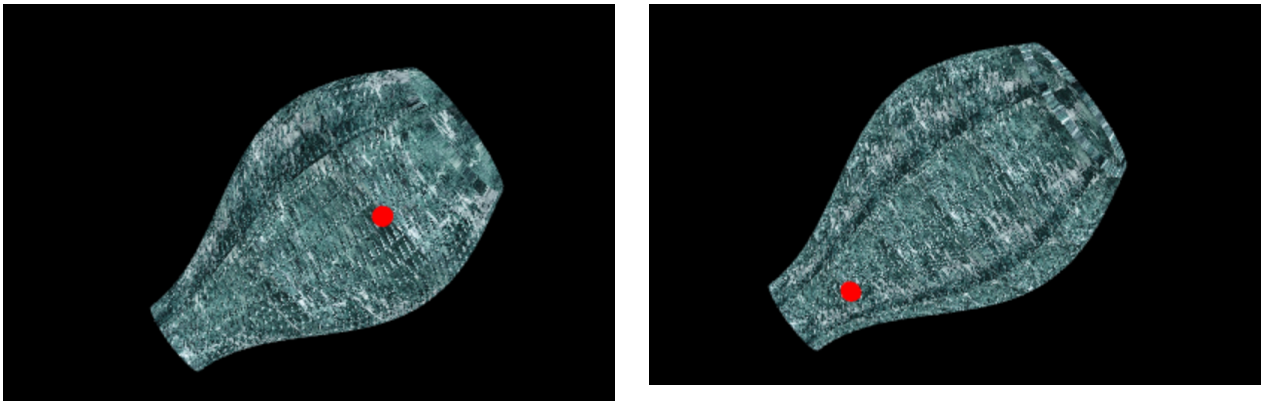

Рис. 4.1-4.2. Переміщення умовної точки

Обертання поверхні в просторі(рис 4.5) та трансформація текстури(рис 4.3-4.4) здійснюються за допомогою відведення миші: необхідно затиснути лівою клавішею миші у області відображення поверхні та потягнути в будь-яку сторону.

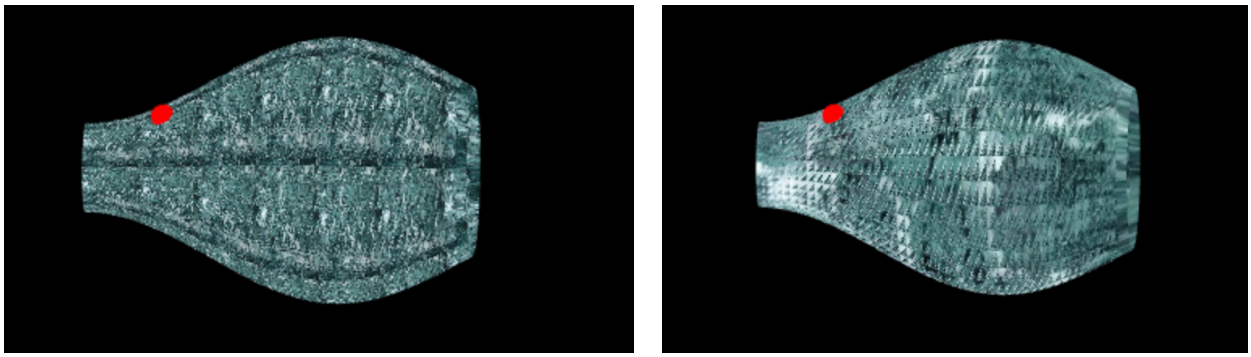

Рис. 4.3-4.4. Трансформація текстури

На рисунку 4.5 можна помітити що точка та текстура залишились на одному і тому самому місці відносно поверхні. Змінилась лише орієнтація поверхні в просторі.

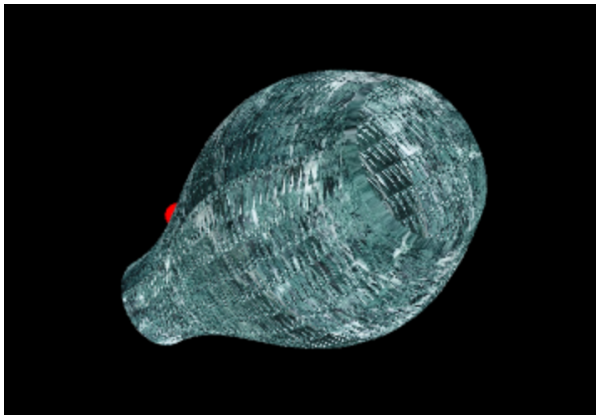

Рис. 4.5. Лише орієнтація поверхні в просторі

5. Код

```
/* Draws a colored cube, along with a set of coordinate axes.
 * (Note that the use of the above drawPrimitive function is not an efficient
 * way to draw with WebGL.  Here, the geometry is so simple that it doesn't matter.)*/
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);
   let para = 3
   let projection = m4.orthographic(-para, para, -para, para, 0, para * 4);

   /* 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);
   gl.uniform1i(shProgram.iTMU, 0);
   gl.enable(gl.TEXTURE_2D);
   gl.uniform1f(shProgram.iScale, userScaleFactor)
   surface.Draw();
   gl.uniform1f(shProgram.iScale, -1.0)
   let a = 2 - 1;
   let c = -2 * Math.PI * a / Math.tan(-0.5);
   let b = 3 * c / 4;
```

```
    let trS = conjugation(map(userPointCoord.x, 0, 1, 0, b), map(userPointCoord.y, 0, 1, 0, Math.PI * 2), a, c)
    gl.uniform3fv(shProgram.iUP, [trS.x, trS.y, trS.z]);
    sphere.DrawPoint();
}
function CreateSurfaceData() {
    let vertexList = [];
    let i = 0;
    let j = 0;
    let a = 2 - 1
    let c = -2 * Math.PI * a / Math.tan(-0.5);
    let b = 3 * c / 4
    while (i < b) {
        while (j < Math.PI * 2) {
            let v1 = conjugation(i, j, a, c)
            let v2 = conjugation(i + 0.2, j, a, c)
            let v3 = conjugation(i, j + 0.2, a, c)
            let v4 = conjugation(i + 0.2, j + 0.2, a, c)
            vertexList.push(v1.x, v1.y, v1.z);
            vertexList.push(v2.x, v2.y, v2.z);
            vertexList.push(v3.x, v3.y, v3.z);
            vertexList.push(v2.x, v2.y, v2.z);
            vertexList.push(v4.x, v4.y, v4.z);
            vertexList.push(v3.x, v3.y, v3.z);
            j += 0.2
        }
        j = 0;
        i += 0.2
    }
    return vertexList;
}
function CreateTextureData() {
    let texCoordList = [];
```

```
    let i = 0;
    let j = 0;
    let a = 2 - 1
    let c = -2 * Math.PI * a / Math.tan(-0.5);
    let b = 3 * c / 4
    while (i < b) {
        while (j < Math.PI * 2) {
            let u = map(i, 0, b, 0, 1);
            let v = map(j, 0, Math.PI * 2, 0, 1);
            texCoordList.push(u, v);
            u = map(i + 0.2, 0, b, 0, 1);
            texCoordList.push(u, v);
            u = map(i, 0, b, 0, 1);
            v = map(j + 0.2, 0, Math.PI * 2, 0, 1);
            texCoordList.push(u, v);
            u = map(i + 0.2, 0, b, 0, 1);
            v = map(j, 0, Math.PI * 2, 0, 1);
            texCoordList.push(u, v);
            u = map(i + 0.2, 0, Math.PI, 0, 1);
            v = map(j + 0.2, 0, Math.PI * 2, 0, 1);
            texCoordList.push(u, v);
            u = map(i, 0, b, 0, 1);
            v = map(j + 0.2, 0, Math.PI * 2, 0, 1);
            texCoordList.push(u, v);
            j += 0.2;
        }
        j = 0
        i += 0.2;
    }
    return texCoordList;
}
```