Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

Розрахунково-графічна робота

Візуалізація графічної та геометричної інформації

Виконав: студент групи ТР-42мп

Хілобок О.О.

Варіант №17

Перевіряв: Каленюк О.С.

Київ – 2024

1 . ВИЗНАЧЕННЯ ЗАВДАННЯ

Презентація завдання:

– поділіться посиланням на репозиторій GIThub із реалізованими завданнями [форма:](https://docs.google.com/forms/d/e/1FAIpQLSfnPxxMWAqfMGMseu7YBJ7zXLIWXIglVatnw1pslh97iTusGA/viewform) (https://docs.google.com/forms/d/e/1FAIpQLSfnPxxMWAqfMGMseu7YBJ7zXLIWXIglVatnw1pslh97iTusGA/viewform) [;](https://docs.google.com/forms/d/e/1FAIpQLSfnPxxMWAqfMGMseu7YBJ7zXLIWXIglVatnw1pslh97iTusGA/viewform)

– переконайтеся, що репозиторій містить роботу, що представляє завдання;

– практичне завдання має знаходитись у філії під назвою CGW.

Операції з координатами текстури – масштабування текстури навколо вказаної користувачем точки.

Вимоги:

– повторне використання відображення текстур із завдання Control;

– реалізувати масштабування текстури (координати текстури) навколо вказаної користувачем точки;

– за допомогою клавіатури має бути можливість переміщати точку по простору поверхні (u, v). Наприклад, клавіші A і D переміщують точку вздовж параметра u, а клавіші W і S переміщують точку вздовж параметра v.

2 . ТЕОРЕТИЧНІ ОСНОВИ

Плоске затінення — це фундаментальна техніка затінення в комп’ютерній графіці, яка використовується для візуалізації 3D-поверхонь рівномірним кольором для кожного багатокутника. У цьому методі кожен багатокутник, як правило, трикутник або чотирикутник, заштрихований на основі одного нормального вектора, який є перпендикулярним до поверхні багатокутника. Це контрастує з методами плавного затінення, такими як затінення Гуро або Фонга, де нормалі інтерполюються по вершинах для створення поступових переходів освітлення та більш реалістичного вигляду.

Основною характеристикою плоского затінення є його здатність чітко окреслювати кожен багатокутник, що призводить до фасетного вигляду, коли окремі грані чітко видно. Це може бути корисним для відтворення малополігональних моделей, архітектурних структур або стилізованої графіки, де потрібна чітка геометрична форма. Крім того, плоске затінення потребує менше обчислень, ніж плавне затінення, оскільки вимагає менше обчислень для освітлення на піксель.

Текстурування є важливою технікою в комп’ютерній графіці, яка покращує візуальну насиченість 3D-моделей, не збільшуючи їх геометричну складність. Він передбачає нанесення 2D-зображення, відомого як текстура, на поверхню 3D-об’єкта. Цей процес додає такі складні деталі, як кольори, візерунки та недоліки поверхні, завдяки чому модель виглядає більш реалістичною та візуально привабливою.

Процес починається з призначення текстурних координат, або UV-координат, кожній вершині 3D-моделі. Ці координати визначають, як двовимірне зображення текстури обертається навколо тривимірної геометрії. Координата U зазвичай представляє горизонтальну вісь, тоді як координата V представляє вертикальну вісь зображення текстури. Шляхом інтерполяції цих координат по поверхні багатокутників кожен фрагмент (або піксель) може точно відібрати відповідний тексель (піксель текстури) із зображення текстури.

Розширені методи текстурування включають використання кількох текстурних карт, таких як дифузні карти для основного кольору, дзеркальні карти для блиску та нормальні карти для імітації деталей поверхні. Наприклад, відображення нормалей спотворює нормалі поверхні на основі текстури, дозволяючи створювати детальні ефекти освітлення без зміни фактичної геометрії. Це покращує відчуття глибини та складності на поверхні, сприяючи більш реалістичному відображенню.

Крім того, методи фільтрації текстур, такі як білінійна та трилінійна фільтрація, використовуються для покращення візуальної якості текстур, коли їх мінімізують або збільшують. Ці методи згладжують зовнішній вигляд текстури, зменшуючи артефакти згладжування та підвищуючи загальну якість зображення.

Масштабування текстури — це техніка, яка регулює розмір текстури, нанесеної на 3D-поверхню. Це дозволяє збільшувати певні області, створювати динамічні візуальні ефекти або виділяти окремі деталі без зміни геометрії моделі. Маніпулюючи U та V координатами текстури, текстури можна рівномірно або нерівномірно масштабувати, пропонуючи різноманітні візуальні результати.

Процес масштабування передбачає зміну координат текстури на основі коефіцієнта масштабування та обраної центральної точки. Коефіцієнт масштабування більше одиниці збільшує текстуру, ефективно збільшуючи масштаб і повторюючи візерунок текстури на поверхні. І навпаки, коефіцієнт, менший за одиницю, зменшує текстуру, поміщаючи її більшу частину в ту саму область. Для масштабування навколо певної точки координати текстури спочатку транслюються так, що центральна точка стає початковою точкою, масштабується, а потім повертається назад. Це локалізоване масштабування підтримує вирівнювання текстури та запобігає спотворенню по всій поверхні.

Обмеження масштабованих координат текстури в діапазоні [0, 1] гарантує, що вибірка текстури залишається в межах зображення текстури, уникаючи артефактів обтікання або розтягування. Масштабування текстури є особливо цінним в інтерактивних програмах, дозволяючи користувачам зосереджуватися на певних областях текстури, наприклад, в інструментах відображення, детальному огляді 3D-моделі або динамічних візуалізаціях, які реагують на введення користувача. Контроль параметрів масштабування текстури в режимі реального часу покращує залучення користувачів і налаштування.

3 . ДЕТАЛІ ВПРОВАДЖЕННЯ

У проекті WebGL CGW керування затіненням здійснюється за допомогою моделі освітлення Phong у фрагментному шейдері, який виконує плавне затінення шляхом інтерполяції нормалей між вершинами. Цей підхід обчислює освітлення для кожного пікселя на основі інтерпольованих нормалей, що призводить до поступових змін освітлення та більш реалістичного вигляду. Хоча поточна реалізація використовує плавне затінення, перехід до плоского затінення включатиме модифікацію вершинного шейдера для передачі однієї нормалі на багатокутник до фрагментного шейдера, усуваючи звичайну інтерполяцію та створюючи фасетний вигляд.

Текстурування реалізується шляхом призначення текстурних координат кожній вершині 3D-поверхні. Вершиний шейдер отримує ці координати через атрибут aTexCoord і передає їх до фрагментного шейдера через варіювання vTexCoord. Фрагментний шейдер потім вибирає розсіяні, дзеркальні та звичайні текстури, використовуючи ці координати, щоб визначити остаточний колір і освітлення кожного пікселя. Це налаштування дозволяє застосовувати кілька текстурних карт для визначення різних властивостей матеріалу, підвищуючи візуальну складність і реалістичність візуалізованої поверхні.

Масштабування текстури досягається шляхом налаштування координат текстури в вершинному шейдері за допомогою уніформ uTextureScale і uStartScalePoint. Шейдер спочатку переводить координати текстури відносно центру масштабування (uStartScalePoint), застосовує коефіцієнт масштабування (uTextureScale), а потім переводить їх назад. Цей процес збільшує або зменшує текстуру навколо вказаної точки, не змінюючи рівномірно всю текстуру. Закріплення гарантує, що масштабовані координати текстури залишаються в допустимому діапазоні [0,1], запобігаючи візуальним артефактам.

Проект містить інтерактивні елементи керування, які дозволяють користувачам переміщувати центр масштабування вздовж простору параметрів (u, v) за допомогою введення з клавіатури. Зокрема, клавіші A і D переміщують точку масштабування вздовж параметра u, тоді як клавіші W і S регулюють її положення вздовж параметра v. Ці вхідні дані оновлюють змінні paramU та paramV, які потім перетворюються на координати текстури та передаються до шейдера. Візуальна червона сфера відображається в центрі масштабування для забезпечення зворотного зв’язку в реальному часі, допомагаючи користувачам зрозуміти точне місце операції масштабування під час маніпулювання параметрами масштабування текстури.

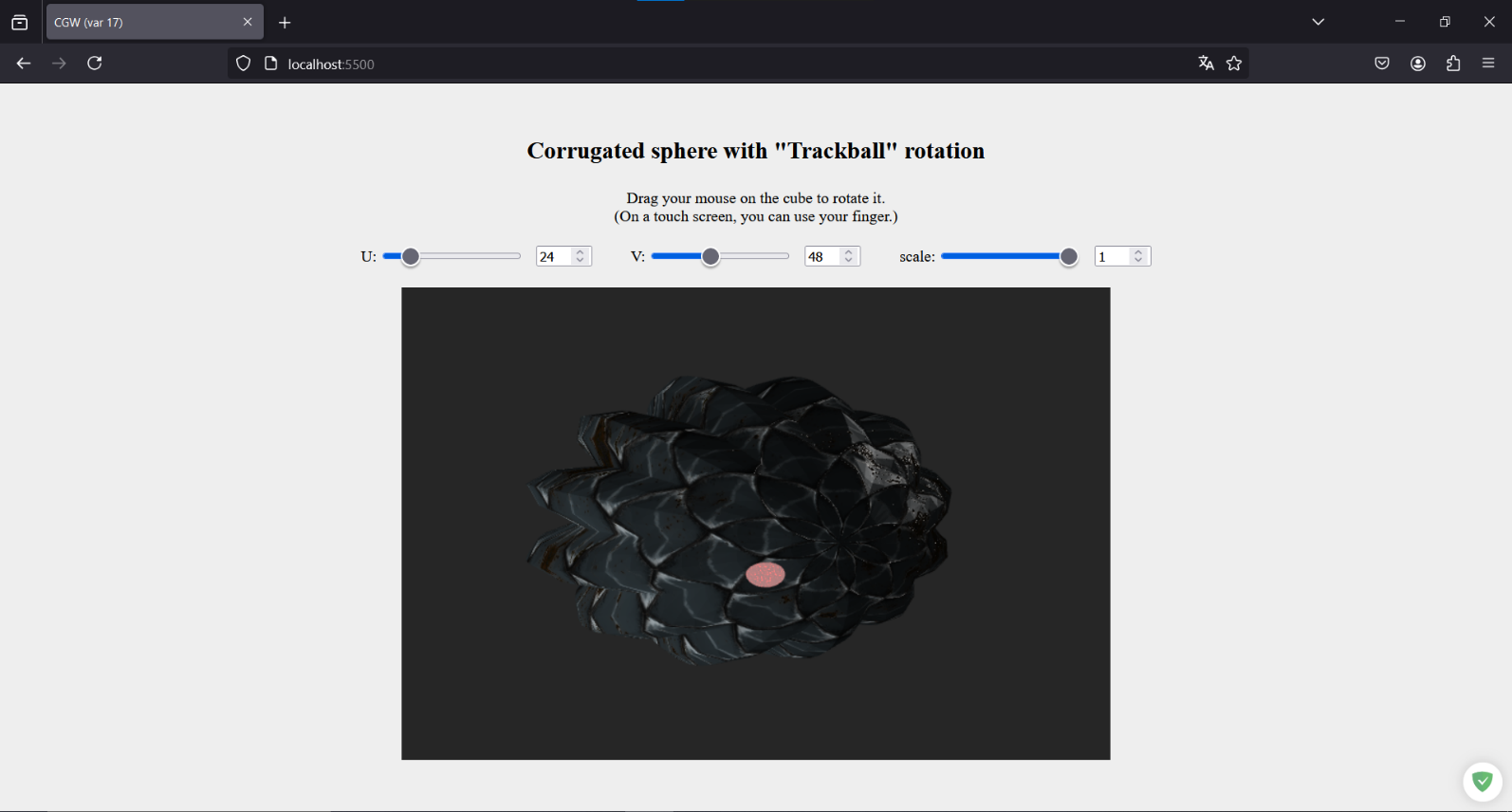
Червона сфера відображається в поточному центрі масштабування на поверхні, пропонуючи миттєвий візуальний зворотний зв’язок користувачам, коли вони налаштовують параметри (u, v). Цей візуальний індикатор допомагає користувачам точно знаходити та керувати областю текстури, яку вони хочуть масштабувати, підвищуючи інтерактивність і зручність використання функції масштабування текстури.

Інтеграція концепцій плоского затінення, комплексного текстурування та динамічного масштабування текстури в рамках проекту WebGL CGW забезпечує надійну структуру для візуалізації детальної та інтерактивної 3D-графіки. Використовуючи шейдерні програми та механізми введення користувачами, проект ефективно керує візуальною складністю та пропонує користувачам інтуїтивно зрозумілий контроль над текстурою.

4 . ІНСТРУКЦІЯ З ВИКОРИСТАННЯ

Щоб використовувати проект WebGL Texture Scaling, рекомендується запускати програму на локальному сервері, щоб усі ресурси завантажувалися правильно. Цього можна легко досягти, наприклад, за допомогою розширення Visual Studio Code Online Server. Після того як сервер активний, доступ до http://localhost:port у веб-браузері запустить програму.

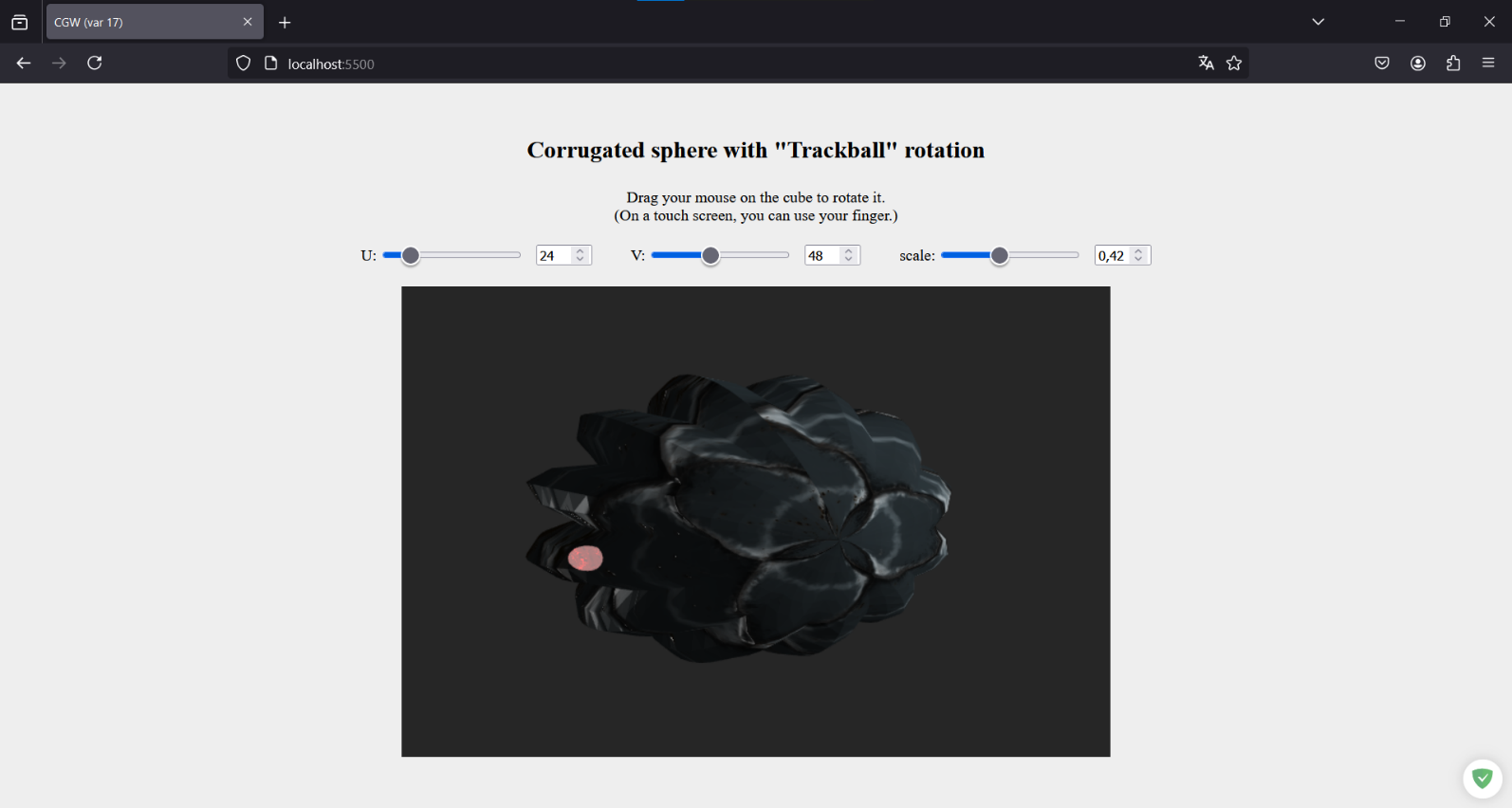
Після запуску (Зображення 4.1) програма відображає 3D-модель із застосованими текстурами. Взаємодія з функцією масштабування текстури полегшується за допомогою елементів керування на клавіатурі: натискання клавіш A і D переміщує центр масштабування ліворуч і праворуч уздовж параметра u, тоді як натискання клавіш W і S зміщує його вгору та вниз уздовж параметра v. Ці вхідні дані змінюють змінні paramU та paramV у реальному часі, що призводить до динамічного оновлення масштабування текстури навколо визначеної точки.



Зображення 4.1 – Приклад роботи програми

Червона сфера відображається на поверхні в поточному центрі масштабування.

Коли параметри масштабування текстури змінюються (Зображення 4.2), текстура на моделі збільшується або зменшується навколо червоної сфери. 3D-модель можна повертати, щоб переглядати ефекти масштабування текстури з різних ракурсів.



Зображення 4.2 – Приклад зміни маштабування

5 ЗРАЗКИ КОДУ

shader.gpu

// Vertex Shader

const vertexShaderSource =

`attribute vec3 aPosition;

attribute vec3 aNormal;

attribute vec3 aTangent;

attribute vec2 aTexCoord;

uniform mat4 uModelViewMatrix;

uniform mat4 uProjectionMatrix;

uniform mat4 uNormalMatrix;

// Additional uniforms for texture scaling

uniform float uTextureScale;

uniform vec2 uStartScalePoint;

// Varying variables to fragment shader

varying vec3 vPosWorld;

varying vec2 vTexCoord;

varying mat3 vTBN;

void main() {

    // 1) World-space position

    vec4 posWorld4 = uModelViewMatrix \* vec4(aPosition, 1.0);

    vPosWorld = posWorld4.xyz;

    // 2) Transform normal

    vec3 N = normalize(mat3(uNormalMatrix) \* aNormal);

    // 3) Transform tangent, orthogonalize

    vec3 T = normalize(mat3(uNormalMatrix) \* aTangent);

    T = normalize(T - dot(T, N) \* N);

    // 4) Bitangent

    vec3 B = cross(N, T);

    // 5) TBN

    vTBN = mat3(T, B, N);

    // 6) Texture scaling around center

    vec2 shifted = aTexCoord - uStartScalePoint;

    shifted \*= uTextureScale;

    shifted += uStartScalePoint;

    vTexCoord = clamp(shifted, 0.0, 1.0); // Clamping to [0,1]

    // 7) Final position

    gl\_Position = uProjectionMatrix \* posWorld4;

}`;

// Fragment Shader

const fragmentShaderSource =

`#ifdef GL\_FRAGMENT\_PRECISION\_HIGH

   precision highp float;

#else

   precision mediump float;

#endif

// Lighting

uniform vec3  uLightPos;

uniform float uAmbientFactor;

uniform float uDiffuseFactor;

uniform float uSpecularFactor;

uniform float uShininess;

// Base color

uniform vec4 uColor;

// Camera direction

uniform vec3 uViewDir;

// Textures

uniform sampler2D uDiffuseSampler;

uniform sampler2D uSpecularSampler;

uniform sampler2D uNormalSampler;

// Varyings

varying vec3 vPosWorld;

varying vec2 vTexCoord;

varying mat3 vTBN;

void main() {

    // 1) Normal from normal map, transform to world space

    vec3 textureNormal = texture2D(uNormalSampler, vTexCoord).rgb;

    textureNormal = 2.0 \* textureNormal - 1.0;

    vec3 N = normalize(vTBN \* textureNormal);

    // 2) Light direction

    vec3 L = normalize(uLightPos - vPosWorld);

    // 3) View direction

    vec3 V = normalize(uViewDir);

    // 4) Reflection vector

    vec3 R = reflect(-L, N);

    // 5) Phong components

    float ambient  = uAmbientFactor;

    float diff     = max(dot(N, L), 0.0);

    float diffuse  = uDiffuseFactor \* diff;

    float specTerm = max(dot(R, V), 0.0);

    float specular = uSpecularFactor \* pow(specTerm, uShininess);

    // Textures

    vec3 diffuseTex  = texture2D(uDiffuseSampler,  vTexCoord).rgb;

    vec3 specularTex = texture2D(uSpecularSampler, vTexCoord).rgb;

    vec3 color = diffuseTex \* diffuse + specularTex \* specular + (ambient \* diffuseTex);

    // Write final pixel color

    gl\_FragColor = vec4(color, 1.0) \* uColor;

}`;