МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра цифрових технологій в енергетиці

Звіт з графічно-розрахункової роботи

з дисципліни «Візуалізація графічної та геометричної інформації»

Виконав:

Студент 1-го курсу магістратури

Інституту атомної та теплової енергетики

Групи ТР-23мп  
Чорний Владислав Олександрович

Прийняв:

Демчишин А. А.

Київ – 2022

**Постановка задачі**

**Варіант:** 27 **-** непарний варіант, реалізувати масштабування.

**Мета роботи:** засвоїти навички роботи з текстурами в WebGL, відобразити текстуру на побудованій поверхні та реалізувати операцію масштабування текстурних координат за варіантом.

**Вимоги:**

* Нанести текстуру на поверхню з практичного завдання №2.
* Реалізувати масштабування текстури (координати текстури) відповідно до визначеної користувачем точки.
* Реалізувати переміщення точку вздовж простору поверхні (u,v) за допомогою клавіатури. Клавіші A і D переміщують точку вздовж параметра u, а клавіші W і S переміщують точку вздовж параметра v.
* Реалізувати зміну значення масштабування використовуючи + для збільшення та — для зменшення.

**Теоретичні відомості**

Текстура (англ. Texture mapping) — це спосіб надання поверхні 3D деталей — полігону: кольору, фактури, блиску, матовості та інших фізичних властивостей (для імітації найчастіше якогось природного матеріалу, наприклад: паперу, дерева, каменю, металу тощо).

Поняття «текстура» є важливим елементом 3D-моделювання, оскільки дозволяє відтворити також малі об'єкти поверхні, створення яких полігонами виявилося б надмірно ресурсомістким. Наприклад, шрами на шкірі, складки на одязі, дрібні камені, предмети на поверхні стін і ґрунту та багато іншого.

Якість поверхні текстури визначається текселями — кількістю пікселів на мінімальну одиницю текстури. Оскільки сама по собі текстура є зображенням, роздільність текстури і її формат відіграють велику роль, яка згодом позначається на загальному враженні від якості графіки у 3D-додатку.

Карта текстури застосовується для утворення певного параметру візуального відображення на поверхні заданої форми. Цей процес нагадує застосування візерунчастого паперу на звичайній білій коробці. Кожній вершині в 3D моделі присвоюється координати текстури (яка у разі 2D відома, як UV координата). Кожна координата текстури є, як мінімум, парою (u,v), яка є горизонтальним і вертикальним розташуванням у просторі текстур відповідно. Значення зазвичай знаходяться в діапазоні [0,1], рисунок 1. Початкова точка (0,0) знаходиться в нижній лівій частині текстури. Для значень (u,v) за межами діапазону [0,1] поведінка текстури визначається додатковими налаштуваннями. Місця відбору зображення згодом інтерполюється по поверхні моделі з отриманням візуального результату. Найчастіше це файл растрового (рідше векторного) зображення з розширенням JPEG, PNG або PAT.

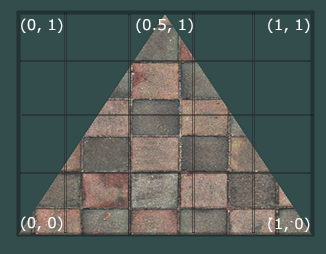


Рисунок 1 — Приклад текстурних координат uv.

Текстури можуть мати 1-3 виміри, хоча 2 виміри найчастіше зустрічаються для видимих поверхонь. Для використання з сучасним апаратним забезпеченням дані карти текстури можуть зберігатися в розрізненому або мозаічному порядку для покращення когерентності кешу. API візуалізації зазвичай керують ресурсами карти текстури (які можуть бути розташовані в пам’яті пристрою) як буферами або поверхнями, і можуть дозволяти «рендеринг до текстури» для додаткових ефектів, таких як постобробка або відображення середовища. Зазвичай вони містять колірні дані RGB (збережені як прямі кольори, стислі формати або індексовані кольори), а іноді й додатковий канал для альфа-змішування (RGBA), особливо для рекламних щитів і накладених текстур наклейок. Можна використовувати альфа-канал (який може бути зручним для зберігання у форматах, аналізованих апаратним забезпеченням) для інших цілей, наприклад для відображення.

Отже, текстура використовується для заповнення поверхонь об'єктів і як шар для додання певного ефекту або зміни геометрії всьому зображенню або його частини

Щоб реалізувати масштабування навколо довільної точки, ми можемо почати зі створення матриці переміщення (translation matrix) T, яка переміщає об'єкт з початкової точки в точку масштабування, та матрицю масштабування R для масштабування об'єкта навколо початкової точки. Тепер для масштабування відповідно точки, ми можемо спершу перемістити точку на місце початкової точки за допомогою інвертування матриці T, записаної як T-1. Потім, ми масштабуємо об'єкт відповідно до початкової точки, за допомогою матриці R, а потім застосовуємо матрицю T для переміщення точки масштабування назад до свого вихідного положення. На рисунку 2 наведено ілюстрацію до кожного з описаних кроків.

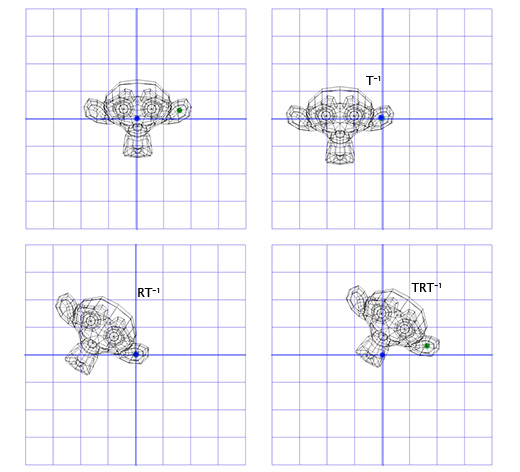


Рисунок 2 — Ілюстрація процесу масштабування відповідно довільно точки.

**Опис деталей реалізації**

Для відображення текстури на поверхні було завантажено зображення за посиланням з інтернету використовуючи базовий інтерфейс JavaScrip. Налаштування параметрів тектсури було встановлено на такі значення: TEXTURE\_WRAP\_S — REPEAT, TEXTURE\_WRAP\_T — REPEAT, TEXTURE\_MIN\_FILTER — LINEAR, TEXTURE\_MAG\_FILTER — LINEAR.

Обрахунок текстурних координат відповідно кожного вертекса реалізовано в основному циклі і здійснюється за допомогою нормалізації параметрів uv, за допомогою яких будується поверхня.

Також, були створені додаткові глобльані параметри які зберігають координати точки масштабування та значення масштабування, які можна змінювати за допомогою вводу з клавіатури. Дані значення надсилаються в шейдер кожного фрейму.

Крім того, в вертексному шейдері було створено функцію яка реалізує масштабування текстурних координат відповідно довільної точки, для мінімізації затрат циклів процесора на обрахевання та відправлення цих даних на графічний процесор.

**Інструкція користувача**

Для взаємодії з програмою реалізовано інтерфей який включає в себе такі клавіші:

* WS – переміщення точки масштабування вздовж параметра v;
* AD – переміщення точки масштабування вздовж параметра u;
* +- – зміна значення масштабування.

Наведемо декілька прикладів нанесеної текстури на поверхню з різними значення параметрів.

На рисунку 3 наведена поверхня з значеннями v і u точки масштабування 0 і 0, значення масштабування дорівнює 1, всі значення за замовчуванням.



Рисунок 3 — Текстура нанесена на поверхню з значеннями за замовчуванням.

На рисунку 4 наведена поверхня з значеннями v і u точки масштабування 45 і 180 відповідно, значення масштабування дорівнює 2.

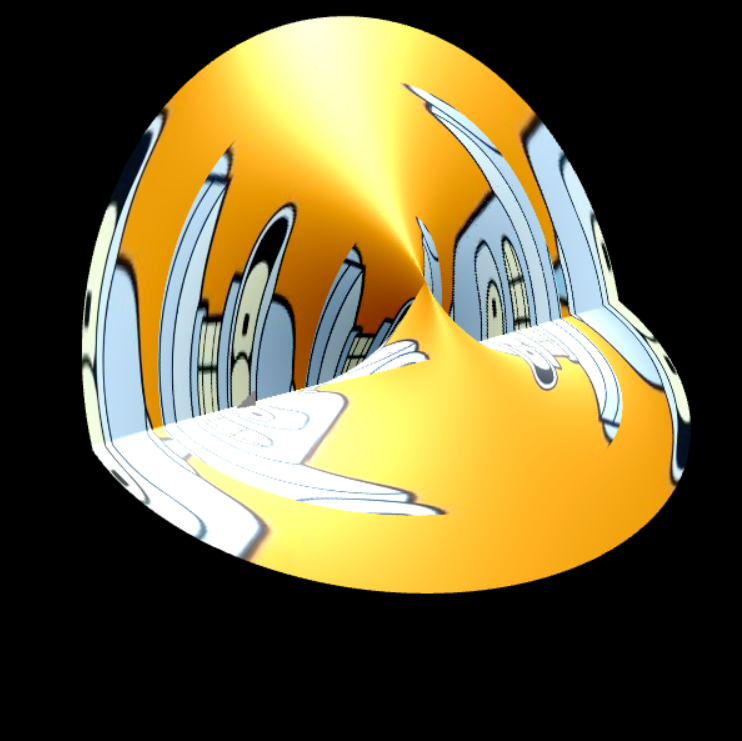


Рисунок 4 — Текстура нанесена на поверхню з довільними значеннями.

**Опис вихідного коду**

В данному розділі описаний основий код реалізації функціоналу та вимог графічно-розрахунковох роботи.

Далі наведемо код функції, яка відповідає за завантаження зображення з інтернету та створення та ініціалізацію зображення, яке далі використовується для відобреження тектсури на поверхні.

function LoadTexture()

{

var texture = gl.createTexture();

gl.bindTexture(gl.TEXTURE\_2D, texture);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_S, gl.REPEAT);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_T, gl.REPEAT);

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, 1, 1, 0, gl.RGBA, gl.UNSIGNED\_BYTE,

new Uint8Array([0, 0, 255, 255]));

var image = new Image();

image.crossOrigin = "anonymous"

image.src = "https://i1.photo.2gis.com/images/profile/30258560049997155\_fe3f.jpg";

image.addEventListener('load', function() {

gl.bindTexture(gl.TEXTURE\_2D, texture);

gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA,gl.UNSIGNED\_BYTE, image);

console.log("Texture is loaded!");

draw();

});

}

Кірм того, наведемо код вертексного та фрагментного шейдерів, які реалізують накладання тексури на поверхню відповідно до текстуринх координат з відповідним масштабуванням текстурних координат відповідно до довільної точки.

// Vertex shader

const vertexShaderSource = `

...

attribute vec2 texcoord;

...

uniform vec2 ScalePointLocation;

uniform float ScaleValue;

...

varying vec2 v\_texcoord;

vec2 ScaleTextCoords(vec2 TexCoords, float Value, vec2 ScalePoint)

{

vec3 CalculatedTextCoords = vec3(TexCoords, 1);

mat3 ScaleMatrix = mat3(vec3(Value, 0.0, 0.0),

vec3(0.0, Value, 0.0),

vec3(0.0, 0.0, 1.0));

mat3 TransformForward = mat3(vec3(1.0, 0.0, 0.0),

vec3(0.0, 1.0, 0.0),

vec3(-ScalePoint.x, -ScalePoint.y, 1.0));

mat3 TransformBackward = mat3(vec3(1.0, 0.0, 0.0),

vec3(0.0, 1.0, 0.0),

vec3(ScalePoint.x, ScalePoint.y, 1.0));

CalculatedTextCoords = TransformForward \* CalculatedTextCoords;

CalculatedTextCoords = ScaleMatrix \* CalculatedTextCoords;

CalculatedTextCoords = TransformBackward \* CalculatedTextCoords;

return CalculatedTextCoords.xy;

}

void main() {

...

v\_texcoord = ScaleTextCoords(texcoord, ScaleValue, ScalePointLocation);

...

}

`;

// Fragment shader

const fragmentShaderSource = `

...

varying vec2 v\_texcoord;

uniform sampler2D u\_texture;

void main() {

...

vec4 TextureColor = texture2D(u\_texture, v\_texcoord);

gl\_FragColor = color;

gl\_FragColor.rgb \*= light;

gl\_FragColor.rgb += specular;

gl\_FragColor += TextureColor;

}`;