МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра цифрових технологій в енергетиці

Варіант - 20

Звіт з графічно-розрахункової роботи з дисципліни «Візуалізація графічної та геометричної інформації»

Виконав:
Студент 1-го курсу магістратури
Групи ТР-21мп
Прачов Віталій

Прийняв:
Демчишин А. А.

Київ – 2022

Розрахунково-графічна робота (Операції над координатами текстури)

Вимоги:

- Нанести текстуру на поверхню з практичного завдання №2.
- Реалізувати масштабування текстури (координати текстури) обертання навколо визначеної користувачем
- Повинна бути можливість переміщати точку 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.

Теоретичні відомості

Нанесення текстури — це техніка, яка використовується в комп'ютерній графіці для нанесення зображення текстури на тривимірний об'єкт або поверхню. Зображення текстури містить інформацію про кольори та візерунки, які мають бути відображені на об'єкті чи поверхні. Ця техніка дозволяє створювати більш реалістичну та детальну 3D-графіку шляхом додавання додаткового шару деталей до поверхні об'єкта.

Існує кілька різних способів застосувати накладання текстури на об'єкт. Одним із поширених методів є використання ультрафіолетового відображення, яке передбачає розгортання 3D-об'єкта в плоске 2D-представлення та призначення координат текстури (координати U та V) кожній вершині об'єкта. Потім зображення текстури наноситься на об'єкт за допомогою цих координат текстури. Інший метод полягає у використанні проекційного відображення, яке передбачає проектування зображення текстури на 3D-об'єкт з певного напрямку або точки зору.

Відображення текстур можна використовувати для різних цілей, зокрема для додавання реалістичних деталей до об'єктів, створення спеціальних ефектів і покращення загального візуального вигляду 3D-сцени. Це важливий метод у сфері комп'ютерної графіки, який широко використовується в різноманітних програмах, включаючи відеоігри, 3D-моделювання та комп'ютерно створені зображення (CGI) у фільмах і на телебаченні.

Опис деталей реалізації

Для того щоб відобразити текстури необхідно спочатку підвантажити зображення, а після чого виконати texImage2d для текстури

Також необхідно задати текстурам параметри:

- TEXTURE_WRAP_S, REPEAT
- TEXTURE_WRAP_T, REPEAT
- TEXTURE_MIN_FILTER, LINEAR
- TEXTURE_MAG_FILTER, LINEAR

Підрахунок координат текстури для кожного вертекса реалізовано в основному циклі і здійснюється за допомогою нормалізації параметрів uv, за допомогою яких будується поверхня. Також додані глобальні параметри які зберігають координати точки масштабування та значення масштабування.

Інструкція користувача

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

Опис вихідного коду

Приклад коду функції яка відповідає за підвантаження зображення та створення текстури з її ініціалізацією зображення

```
function LoadImage(texture) {
  const image = new Image();
  image.src = 'texture.png';
  image.addEventListener( type: 'load',  listener: function () {
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
    draw();
  });
}
```

```
function LoadTexture() {
  const 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( elements: [0, 0, 255, 255]));
  LoadImage(texture)
}
```

Код вертексного та фрагментного шейдера

```
vec2 ScaleTextCoords(vec2 TextCoords, float Value, vec2 ScalePoint)
{
    vec3 CalculatedTextCoords = vec3(TextCoords, 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() {
    if (bDrawpoint == true) {
        gl_Position = ModelViewProjectionMatrix * vec4(ScalePointWorldLocation, 1.0);
        gl_PointSize = 30.0;
    } else {
        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_textcoord = ScaleTextCoords(textcoord, ScaleValue, ScalePointLocation);
        gl_PointSize = 1.0;
    }
}
```

```
void main() {
    if (bDrawpoint == true) {
        gl_FragColor = color;
    } else {
        vec3 normal = normalize(v_normal);
        vec3 surfaceToLightDirection = normalize(v_surfaceToLight);
        vec3 surfaceToViewDirection = normalize(v_surfaceToView);
        vec3 halfVector = normalize(surfaceToLightDirection + surfaceToViewDirection);
        float shininess = 4.0;
        float innerLimit = cos(5.0 * 3.1415 / 180.0);
        float outerLimit = cos(15.0 * 3.1415 / 180.0);
        float dotFromDirection = dot(surfaceToLightDirection, -LightDirection);
        float inLight = smoothstep(outerLimit, innerLimit, dotFromDirection);
        float light = inLight * dot(normal, surfaceToLightDirection);
        float specular = inLight * pow(dot(normal, halfVector), shininess);
        vec4 TextureColor = texture2D(u_texture, v_textcoord);
        gl_FragColor = color;
        gl_FragColor.rgb *= light;
        gl_FragColor.rgb += specular;

        gl_FragColor += TextureColor;
    }
}
```