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

КОНТРОЛЬНА РОБОТА №2

Виконав:
Студент групи ІП-22
Підпанюк В.А.

Перевірив:
доц. каф. ІПІ
Родіонов П.Ю.

Київ 2024

КОНТРОЛЬНА РОБОТА №2

З ДИСЦИПЛІНИ

«КОМП’ЮТЕРНА ГРАФІКА ТА МУЛЬТИМЕДІА»

Завдання:

1. Написати програму, що виводить на екран зображення першої літери Вашого прізвища.
2. Застосувати довільні кольори та текстуру до створеного об'єкту.
3. Реалізувати модель освітлення для графічної сцени.
4. Написати коментарі до логічних блоків програмного коду.
5. Оформити результати виконання роботи у відповідності до вимог.

Проектування фігури із анімацією та освітленням

У ході даної роботи я моделював літеру “П” (Підпанюк), за допомогою програмного засобу WebGL було додано текстуру “камінь”, анімацію обертання, та освітлення для нашої фігури (рис. 1.1).

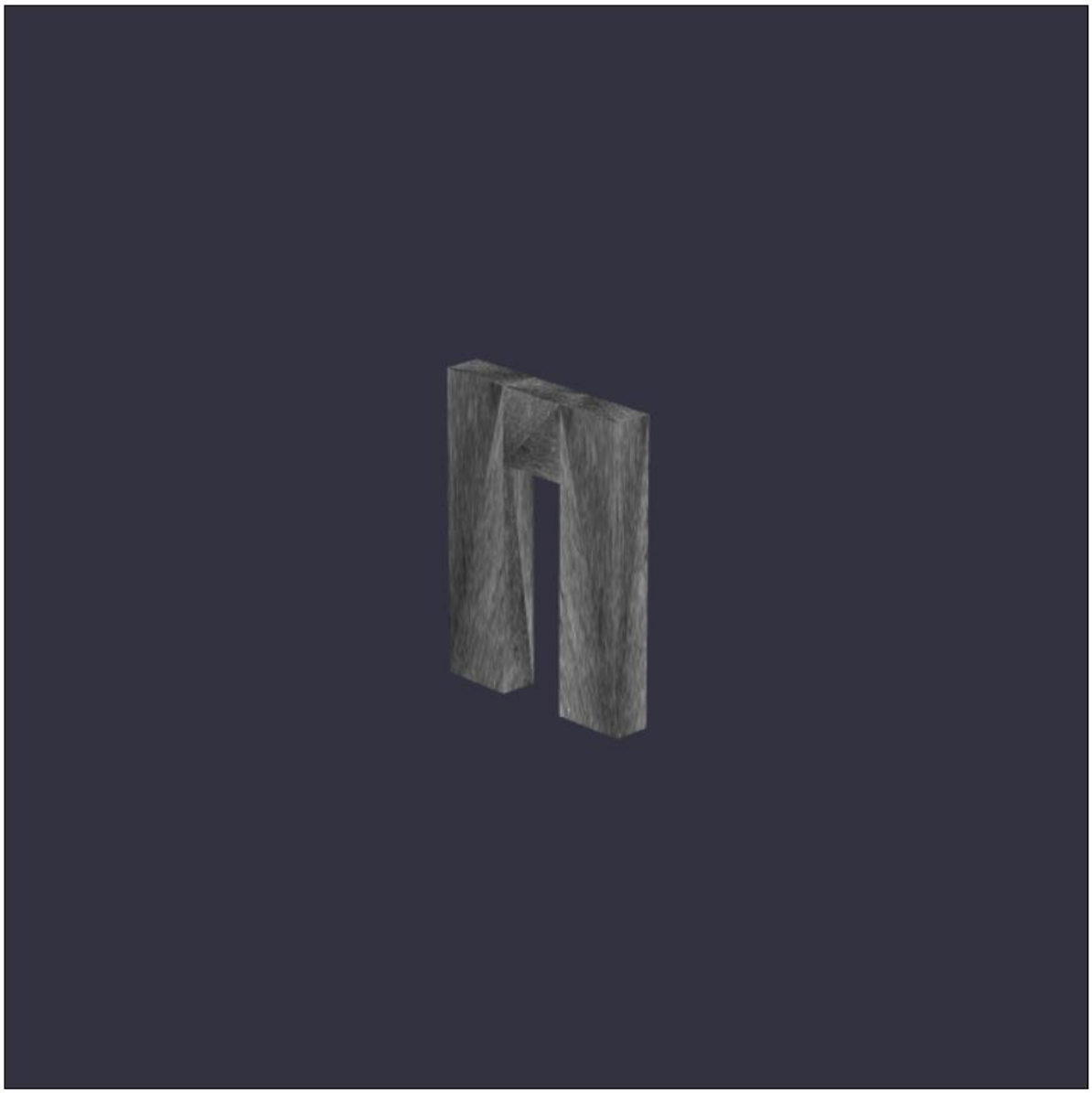

рис. 1.1. Фігура із освітленням спереду

Висновок:

У ході даної роботи я створив 3D-модель, що представляє першу літеру мого прізвища, використовуючи WebGL. Я застосував довільні кольори та текстури до об'єкта, а також реалізував модель освітлення для створення більш реалістичного вигляду сцени. Освітлення включає дифузне та амбієнтне освітлення для досягнення необхідного ефекту. Усі важливі етапи програми були прокоментовані для зрозумілості логіки роботи.

ПРОГРАМНИЙ КОД

HTML:

```
<!DOCTYPE HTML>
<html>
   <head>
      <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>KR_2</title>
      <style>
         body, html { margin: 0; padding: 0; overflow: hidden; }
         canvas { display: block; border: 1px solid black; margin: 3%;}
      </style>
   </head>

   <body>
      <canvas id = "mycanvas" width = "700" height = "700"></canvas>
      <script src="https://cdn.jsdelivr.net/npm/gl-matrix@2.8.1/dist/gl-matrix-
min.js"></script>
      <!-- <script src="scripts/perspective.js"></script> -->
      <script src="scripts/V_drawing.js"></script>
   </body>
</html>
```

JS:

```
const canvas = document.getElementById('mycanvas'); const gl =
canvas.getContext('webgl');
if (!gl) { alert("WebGL не підтримується в цьому браузері"); }
gl.disable(gl.CULL_FACE);
const vertexShaderSource = ` attribute vec4 a_position; attribute vec4 a_color;
attribute vec3 a_normal; // Додано нормалі attribute vec2 a_texcoord;
uniform mat4 u_matrix; uniform mat4 u_normalMatrix; // Для перетворення
нормалей в світовий простір
varying vec4 v_color; varying vec2 v_texcoord; varying vec3 v_normal; //
Передаємо нормаль у фрагментний шейдер varying vec3 v_position; //
Позиція для розрахунків освітлення
void main() { gl_Position = u_matrix * a_position; v_color = a_color;
v_texcoord = a_texcoord; v_normal = mat3(u_normalMatrix) * a_normal; //
```

Трансформуємо нормаль v_position = (u_matrix * a_position).xyz; // Позиція для розрахунків освітлення }
`;
const fragmentShaderSource = ` precision mediump float;
varying vec4 v_color; varying vec2 v_texcoord; varying vec3 v_normal;
varying vec3 v_position;
uniform sampler2D u_texture; uniform vec3 u_lightPosition; // Позиція джерела світла uniform vec3 u_lightColor; // Колір світла uniform vec3 u_ambientLight; // Амбієнтне світло
void main() { // Нормалізація нормалі vec3 normal = normalize(v_normal);
// Вектор до джерела світла
vec3 lightDir = normalize(u_lightPosition - v_position);

// Розрахунок дифузного освітлення
float diff = max(dot(normal, lightDir), 0.0);
vec3 diffuse = u_lightColor * diff;

// Амбієнтне освітлення
vec3 ambient = u_ambientLight;

// Підсумковий колір
vec4 texColor = texture2D(u_texture, v_texcoord);
vec3 lighting = ambient + diffuse;

gl_FragColor = vec4(texColor.rgb * lighting, texColor.a);

}
`;
function createShader(gl, type, source) { const shader = gl.createShader(type); gl.shaderSource(shader, source); gl.compileShader(shader); const success = gl.getShaderParameter(shader, gl.COMPILE_STATUS); if (success) { return shader; } console.log(gl.getShaderInfoLog(shader)); gl.deleteShader(shader); }
function createProgram(gl, vertexShader, fragmentShader) { const program = gl.createProgram(); gl.attachShader(program, vertexShader); gl.attachShader(program, fragmentShader); gl.linkProgram(program); const success = gl.getProgramParameter(program, gl.LINK_STATUS); if (success) { return program; } console.log(gl.getProgramInfoLog(program)); gl.deleteProgram(program); }

```
const vertexShader = createShader(gl, gl.VERTEX_SHADER,
vertexShaderSource); const fragmentShader = createShader(gl,
gl.FRAGMENT_SHADER, fragmentShaderSource); const program =
createProgram(gl, vertexShader, fragmentShader);
const positions = [ // 1 -0.06, 0.08, 0.01, -0.06, 0.08, -0.01, -0.02, 0.08, -0.01, -
0.06, 0.08, 0.01, -0.02, 0.08, 0.01, -0.02, 0.08, -0.01,
-0.02, -0.08, -0.01,
-0.02, 0.08, 0.01,
-0.02, 0.08, -0.01,
-0.02, 0.08, 0.01,
-0.02, -0.08, 0.01,
-0.02, -0.08, -0.01,

-0.06, 0.08, -0.01,
-0.02, 0.08, -0.01,
-0.02, -0.08, -0.01,
-0.06, -0.08, -0.01,
-0.06, 0.08, -0.01,
-0.02, -0.08, -0.01,

-0.06, -0.08, -0.01,
-0.06, -0.08, 0.01,
-0.02, -0.08, -0.01,
-0.02, -0.08, 0.01,
-0.06, -0.08, 0.01,
-0.02, -0.08, -0.01,

-0.02, 0.08, 0.01,
-0.02, -0.08, 0.01,
-0.06, -0.08, 0.01,
-0.02, 0.08, 0.01,
-0.06, 0.08, 0.01,
-0.06, -0.08, 0.01,

-0.06, -0.08, -0.01,
-0.06, 0.08, -0.01,
-0.06, -0.08, 0.01,
-0.06, 0.08, -0.01,
```

```
-0.06, 0.08, 0.01,
-0.06, -0.08, 0.01,

// 2
-0.02, 0.08, 0.01,
-0.02, 0.08, -0.01,
0.02, 0.08, -0.01,
-0.02, 0.08, 0.01,
0.02, 0.08, -0.01,
0.02, 0.08, 0.01,
-0.02, 0.08, 0.01,

0.02, 0.08, 0.01,
-0.02, 0.04, 0.01,
0.02, 0.08, 0.01,
0.02, 0.04, 0.01,
-0.02, 0.04, 0.01,
0.02, 0.08, 0.01,

0.02, 0.04, -0.01,
0.02, 0.04, 0.01,
-0.02, 0.04, 0.01,
0.02, 0.04, -0.01,
0.02, 0.04, 0.01,
-0.02, 0.04, 0.01,
-0.02, 0.04, -0.01,
0.02, 0.04, -0.01,
-0.02, 0.04, 0.01,

-0.02, 0.04, -0.01,
-0.02, 0.04, -0.01,
0.02, 0.04, -0.01,
0.02, 0.08, -0.01,
-0.02, 0.04, -0.01,
-0.02, 0.08, -0.01,
0.02, 0.08, -0.01,
-0.02, 0.04, -0.01,
```

```
// 3
0.06, 0.08, 0.01,
0.06, 0.08, -0.01,
0.02, 0.08, -0.01,
0.06, 0.08, 0.01,
0.02, 0.08, 0.01,
0.02, 0.08, -0.01,

0.02, -0.08, -0.01,
0.02, 0.08, 0.01,
0.02, 0.08, -0.01,
0.02, 0.08, 0.01,
0.02, -0.08, 0.01,
0.02, -0.08, -0.01,

0.06, 0.08, -0.01,
0.02, 0.08, -0.01,
0.02, -0.08, -0.01,
0.06, -0.08, -0.01,
0.06, 0.08, -0.01,
0.02, -0.08, -0.01,

0.06, -0.08, -0.01,
0.06, -0.08, 0.01,
0.02, -0.08, -0.01,
0.02, -0.08, 0.01,
0.06, -0.08, 0.01,
0.02, -0.08, -0.01,

0.02, 0.08, 0.01,
0.02, -0.08, 0.01,
0.06, -0.08, 0.01,
0.02, 0.08, 0.01,
0.06, 0.08, 0.01,
0.06, -0.08, 0.01,

0.06, -0.08, -0.01,
0.06, 0.08, -0.01,
```

```
0.06, -0.08, 0.01,
0.06, 0.08, -0.01,
0.06, 0.08, 0.01,
0.06, -0.08, 0.01,

];
const faceColors = new Array(128).fill([1.0, 0.0, 0.0, 1.0]).flat();
const linePositions = [ -0.2, -0.2, 0.2, 0.2, -0.2, 0.2, 0.2, 0.2, 0.2, -0.2, 0.2, 0.2,
-0.2, -0.2, -0.2,
 0.2, -0.2, -0.2,
 0.2,  0.2, -0.2,
-0.2,  0.2, -0.2,

-0.2, -0.2,  0.2,
-0.2, -0.2, -0.2,
 0.2, -0.2,  0.2,
 0.2, -0.2, -0.2,
 0.2,  0.2,  0.2,
 0.2,  0.2, -0.2,
-0.2,  0.2,  0.2,
-0.2,  0.2, -0.2,

];
const edgeColors = new Array(24).fill([0.0, 0.0, 0.0, 1.0]).flat();
const positionBuffer = gl.createBuffer(); gl.bindBuffer(gl.ARRAY_BUFFER,
positionBuffer); gl.bufferData(gl.ARRAY_BUFFER, new
Float32Array(positions), gl.STATIC_DRAW);
const colorBuffer = gl.createBuffer(); gl.bindBuffer(gl.ARRAY_BUFFER,
colorBuffer); gl.bufferData(gl.ARRAY_BUFFER, new
Float32Array(faceColors), gl.STATIC_DRAW);
function setAttribute(buffer, attribute, size) {
gl.bindBuffer(gl.ARRAY_BUFFER, buffer); const location =
gl.getAttribLocation(program, attribute); gl.enableVertexAttribArray(location);
gl.vertexAttribPointer(location, size, gl.FLOAT, false, 0, 0); setTexcoords(gl); }
const normalBuffer = gl.createBuffer(); gl.bindBuffer(gl.ARRAY_BUFFER,
normalBuffer);
const normals = [
```

```
0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,

0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1,

0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,

0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0,

-1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0,

1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,

];
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(normals),
gl.STATIC_DRAW);
function setTexcoords(gl) { const texcoordBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, texcoordBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([ 0, 0, 1, 0, 0, 1, 0, 1, 1,
0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0,
0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1,
0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0,
0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1,
0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0,
0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1,
0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1 ]), gl.STATIC_DRAW);
const texcoordLocation = gl.getAttribLocation(program, 'a_texcoord');
gl.enableVertexAttribArray(texcoordLocation);
gl.vertexAttribPointer(texcoordLocation, 2, gl.FLOAT, false, 0, 0);

const normalLocation = gl.getAttribLocation(program, 'a_normal');
gl.enableVertexAttribArray(normalLocation);
gl.vertexAttribPointer(normalLocation, 3, gl.FLOAT, false, 0, 0);

}
```

```
const fieldOfViewRadians = Math.PI / 12; const aspect = canvas.width / canvas.height; const zNear = 0.1; const zFar = 100.0; const projectionMatrix = mat4.create(); mat4.perspective(projectionMatrix, fieldOfViewRadians, aspect, zNear, zFar);
const texture = gl.createTexture(); const image = new Image(); image.onload = () => { gl.bindTexture(gl.TEXTURE_2D, texture); gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image); gl.generateMipmap(gl.TEXTURE_2D); };
image.src = '/textures/download-resizehood.com (2).png';
if(image == null) { console.log.error("Image is null!"); } else { console.log("all is ok!"); }
image.addEventListener('load', function() { gl.bindTexture(gl.TEXTURE_2D, texture); gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,gl.UNSIGNED_BYTE, image); gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR); gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE); gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
gl.generateMipmap(gl.TEXTURE_2D); });
let rotationY = 0; function drawScene() { gl.clearColor(0.2, 0.2, 0.25, 1.0); gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT); gl.enable(gl.DEPTH_TEST);

rotationY += 0.01;

const modelMatrix = mat4.create();
mat4.translate(modelMatrix, modelMatrix, [0, 0, -2]);
mat4.rotateX(modelMatrix, modelMatrix, Math.PI / 9);
mat4.rotateY(modelMatrix, modelMatrix, rotationY);

const mvpMatrix = mat4.create();
mat4.multiply(mvpMatrix, projectionMatrix, modelMatrix);

const normalMatrix = mat4.create();
mat4.invert(normalMatrix, modelMatrix);
mat4.transpose(normalMatrix, normalMatrix);
```

```
gl.uniformMatrix4fv(gl.getUniformLocation(program, 'u_normalMatrix'), false,
normalMatrix);
gl.uniform3f(gl.getUniformLocation(program, 'u_lightPosition'), 0.0, 0.0, 5.0);
gl.uniform3f(gl.getUniformLocation(program, 'u_lightColor'), 1.0, 1.0, 1.0);
gl.uniform3f(gl.getUniformLocation(program, 'u_ambientLight'), 0.1, 0.1, 0.1);

gl.useProgram(program);
setAttribute(positionBuffer, 'a_position', 3);
setAttribute(colorBuffer, 'a_color', 4);
gl.uniformMatrix4fv(gl.getUniformLocation(program, 'u_matrix'), false,
mvpMatrix);
gl.drawArrays(gl.TRIANGLES, 0, positions.length / 3);

requestAnimationFrame(drawScene);

}
requestAnimationFrame(drawScene);
```