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

ВІЗУАЛІЗАЦІЯ ГРАФІЧНОЇ ТА ГЕОМЕТРИЧНОЇ ІНФОРМАЦІЇ

Розрахунково-графічна робота
Варіант №4

Виконав:
студент 1-го курсу НН ІАТЕ
групи ТР-31мп
Вербіцький Євген Степанович

Перевірив: Демчишин А. А.

Київ – 2023

ЗАВДАННЯ

1. Нанести текстуру на поверхню з практичного завдання №2;

2. Реалізувати обертання текстури (координат текстури) навколо вказаної користувачем точки;

3. Реалізувати можливість переміщення точки 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.

ТЕОРЕТИЧНІ ОСНОВИ

WebGL (Web Graphics Library) — це технологія, яка дозволяє створювати 3D-графіку в браузері без необхідності встановлення додаткових плагінів чи розширень. Вона базується на OpenGL ES (Embedded Systems), із рядом доповнень для взаємодії з елементами веб-сторінки. За допомогою WebGL розробники можуть створювати візуально багаті та інтерактивні веб-додатки, ігри та візуалізації, використовуючи високоякісну 3D-графіку.

Текстура – це зображення, яке може бути призначено 3D-моделі, щоб надати їй додаткову деталізацію, колір, рельєф або інші візуальні характеристики. Текстури використовуються для імітації реальних матеріалів, створення візуальних ефектів та покращення реалістичності сцени.

Основна ідея використання текстур полягає в тому, щоб забезпечити 3D-об'єкти додатковим деталізованим виглядом, не збільшуючи кількість полігонів в моделі. Це дозволяє оптимізувати продуктивність, забезпечуючи водночас високу якість візуалізації.

Текстури створюється в WebGL за допомогою функції createTexture(). Далі ініціалізується об’єкт Image, в якому вказується URL-адреса зображення, що й буде використовуватись як текстура. У WebGL кожен вершинний піксель на 3D-моделі має відповідні координати текстури, які вказують, який фрагмент текстури використовувати для цього пікселя. Для оптимізації використання текстур можна застосовувати різні методи фільтрації, такі як білінійна фільтрація або трилінійна фільтрація.

Існує декілька видів текстур:

- 2D-текстури – найбільш поширені текстури, які використовуються для більшості задач;
- текстури висот (heightmaps) – використовуються для створення рельєфу на поверхні 3D-об'єктів;
- куб-текстури – використовуються для створення оточення або інших візуальних ефектів, які вимагають 360-градусного зображення.

Загалом, текстури є важливою частиною графічного програмування та візуалізації. Вони дозволяють створювати реалістичні сцени, оптимізуючи при цьому продуктивність та використання ресурсів. Завдяки текстурам можна досягти високої якості візуалізації без значних втрат у продуктивності.

Обертання в графіці – це процес зміни орієнтації об'єкта або сцени навколо певної осі або точки. Це одна з фундаментальних операцій у 3D-графіці, яка дозволяє створювати візуальні ефекти, змінювати перспективу та взаємодію об'єктів на сцені.

Для реалізації обертання в 3D-графіці використовуються математичні формули, зокрема матриці обертання, які описують трансформації координат об'єктів у просторі. У математичній моделі обертання використовуються три головні осі: X, Y та Z.

Для реалізації обертання навколо довільної точки застосовується техніка зсуву. В такому випадку, текстурна координата переноситься на початок координат, навколо яких відбувається обертання на заданий кут шляхом застосування відповідної матриці обертання. Після цього, отримані текстурні координати переносяться назад.

РЕАЛІЗАЦІЯ

Координати текстур зберігаються в створеному для цього буфері:

```
gl.bindBuffer(gl.ARRAY_BUFFER, this.iTextCoordBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textCoords), gl.STREAM_DRAW);
```

Збережені координати використовуються під час рендерингу сцени:

```
gl.bindBuffer(gl.ARRAY_BUFFER, this.iTextCoordBuffer);
gl.vertexAttribPointer(shProgram.iAttribTextCoord, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(shProgram.iAttribTextCoord);
```

Для встановлення відповідності між текстурними координатами та вершинами поверхні використовуються параметри r та theta. Для цього значення параметрів необхідно нормалізувати до проміжку 0..1:

```
function CalculateTextCoord(r, theta) {
  r = (r - 0.25)/(maxR - 0.25);
  theta = theta / 2*Math.PI;

  return { r, theta };
}
```

Ця функція викликається для кожної вершини поверхні, яку ми знаходимо в функції CreateSurfaceData(), і отримані відповідні текстурні координати заносяться в масив textCoordList. Саме цей масив заповнює буфер координат текстур.

Зображення, яке й буде текстурою, завантажується в функції LoadTexture(). Для цього використовується об’єкт класу Image, в якому вказується URL-посилання на віддалений ресурс, де зберігається потрібне зображення. Після вдалого завантаження, зображення прив’язується до створеного в цій же функції об’єкту текстури. Окрім цього, вказуються параметри фільтрації текстури. В кінці викликається функція рендерингу сцени:

```
function LoadTexture() {

  var texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
```

```
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    var image = new Image();
    image.crossOrigin = "anonymous";
    image.src = "https://i.ibb.co/1TgPH2f/texture-1.jpg";
    image.addEventListener('load', () => {
      gl.bindTexture(gl.TEXTURE_2D, texture);
      gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
      draw();
    }
    );
}
```

Для обертання текстури модифікували вершинний шейдер, додавши в нього дві функції, що відповідають за створення матриці обертання та зсуву:

```
mat4 rotate(float angleInRadians) {
    float c = cos(angleInRadians);
    float s = sin(angleInRadians);
    return mat4(
        c,   s, 0.0, 0.0,
       -s,   c, 0.0, 0.0,
       0.0, 0.0, 1.0, 0.0,
       0.0, 0.0, 0.0, 1.0
    );
}
mat4 translate(float tx, float ty) {
    return mat4(
       1.0, 0.0, 0.0, 0.0,
       0.0, 1.0, 0.0, 0.0,
       0.0, 0.0, 1.0, 0.0,
       tx,  ty,  0.0, 1.0
    );
}
```

Процес обертання текстурної координати реалізований наступним чином:

1. знаходимо матриці обертання та зсуву

```
mat4 rotateMat = rotate(angleInRadians);
mat4 translateMat = translate(-userPoint.x, -userPoint.y);
mat4 translateMatBack = translate(userPoint.x, userPoint.y);
```

2. Переносимо текстурну координату до початку координат

```
vec4 textCoordTr = translateMat*vec4(textCoord,0,1.0);
```

3. Обертаємо координату

```
vec4 textCoordRotated = rotateMat*textCoordTr;
```

4. Повертаємо координату назад та передаємо отримане значення в фрагментний шейдер

```
vec4 textCoordTrBack = translateMatBack*textCoordRotated;
textInterp = textCoordTrBack.xy;
```

Функція handleKeyPress() реалізовує можливість переміщення точки на поверхні, навколо якої обертається текстура, за допомогою клавіш WASD. W та S переміщують точку по параметру r, A та D – по параметру theta:

```
function handleKeyPress(event) {
  let stepSize = 0.05;
  switch (event.key) {
    case 'w': case 'W':
      userPoint[0] += stepSize;
      if (userPoint[0] > maxR) {
        userPoint[0] = 0.25;
      }
      break;
    case 's': case 'S':
      userPoint[0] -= stepSize;
      if (userPoint[0] < 0.25) {
        userPoint[0] = maxR;
      }
      break;
    case 'a': case 'A':
      userPoint[1] -= stepSize;
      if (userPoint[1] < 0) {
        userPoint[1] = 2 * Math.PI;;
      }
      break;
    case 'd': case 'D':
      userPoint[1] += stepSize;
      if (userPoint[1] > 2 * Math.PI) {
        userPoint[1] = 0;
      }
      break;
    default: return;
  }
  updateSurface();
}
```

ІНСТРУКЦІЯ КОРИСТУВАЧА

Після запуску програми бачимо поверхню з нанесеною на неї текстурою (рис. 1).

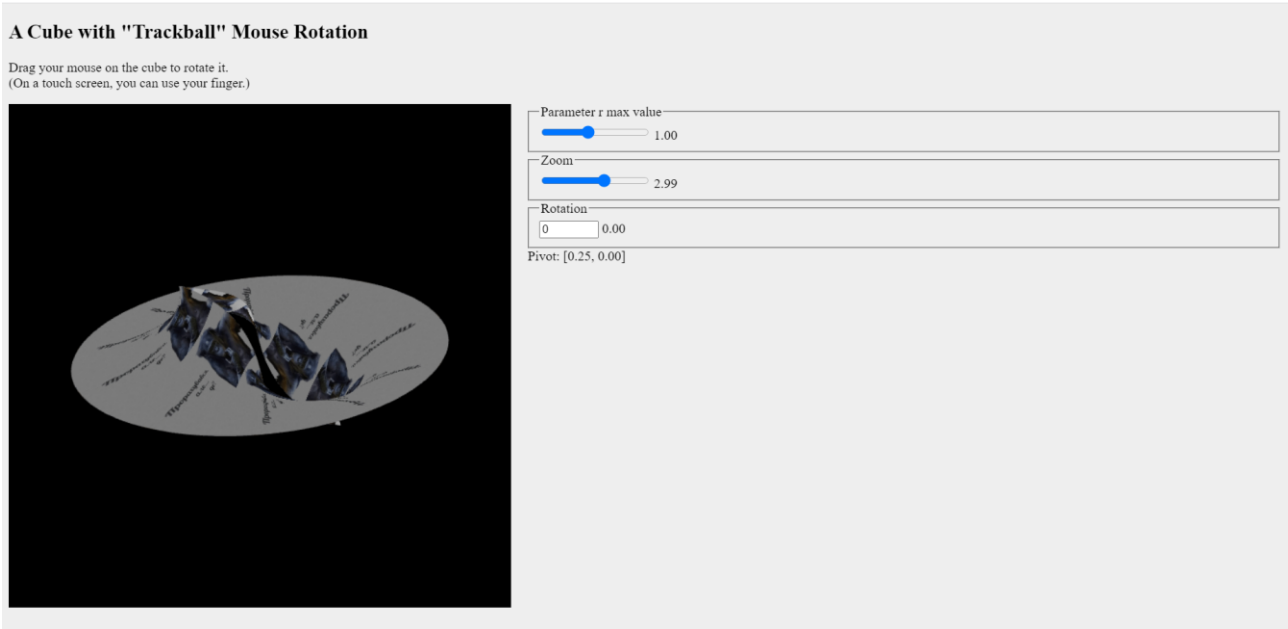

Рисунок 1. – Поверхня з параметрами за замовчуванням

Вказуючи в полі Rotation значення, ми можемо обертати текстуру на відповідний кут. На рисунку 2 текстура обернена на 90 градусів.

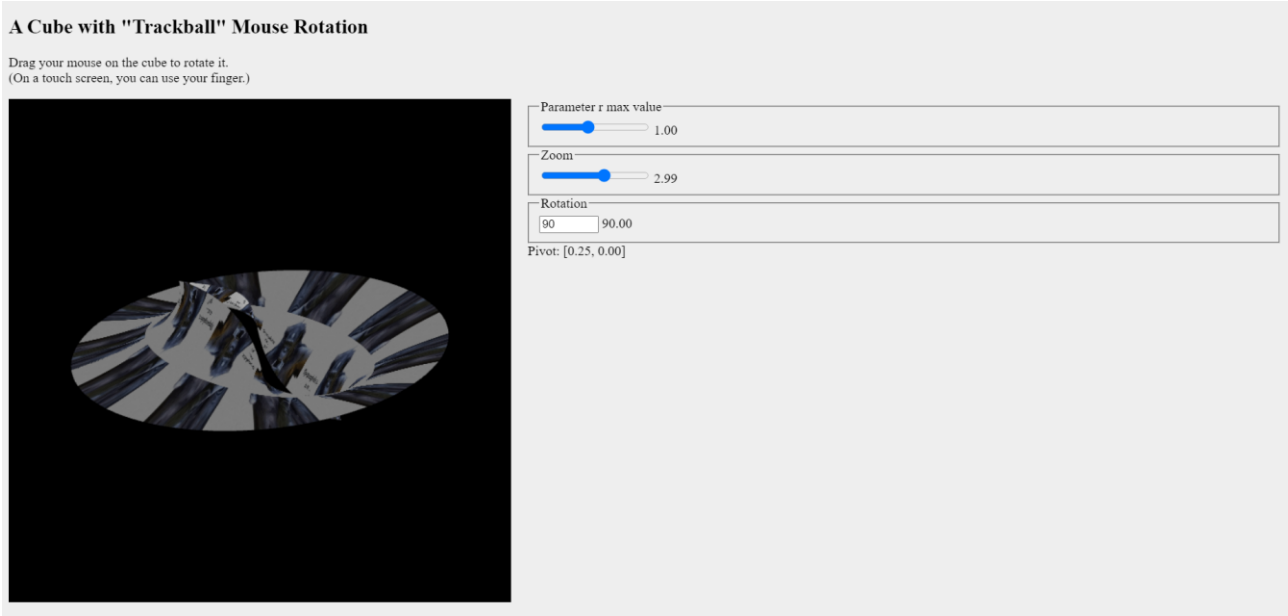

Рисунок 2. – Обертання текстури на 90 градусів

Використовуючи клавіші W, A, S, D, ми можемо переміщувати точку на нашій поверхні, відносно якої буде обертатися текстура. Координати точки обертання представлені в полі Pivot. На рисунку 3 представлена обернена на 90 градусів текстура відносно точки (1.0, 2.0).

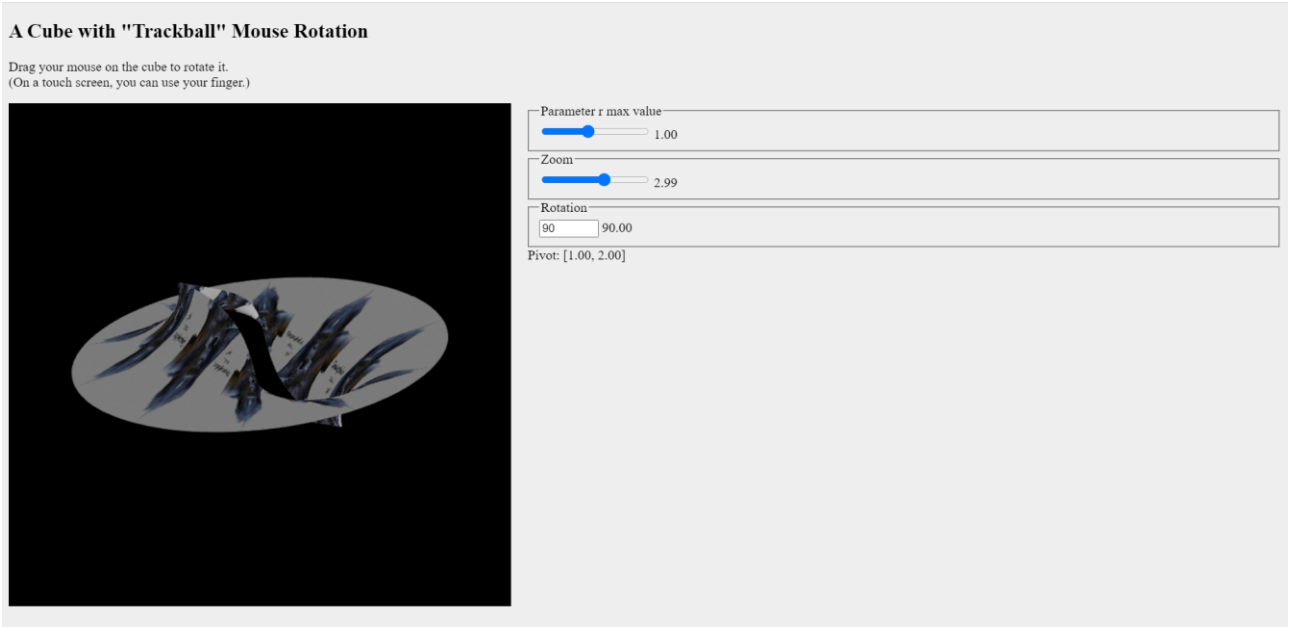

Рисунок 3. – Обертання текстури на 90 градусів навколо точки (1.0, 2.0)

Окрім цього, програма дозволяє змінювати максимальне значення параметру r, яке впливає на вигляд поверхні, а також надає можливість змінювати масштабування.

ВИХІДНИЙ КОД

```
function Model(name) {
    this.name = name;
    this.iVertexBuffer = gl.createBuffer();
    this.iNormalBuffer = gl.createBuffer();
    this.iTextCoordBuffer = gl.createBuffer();
    this.count = 0;

    this.BufferData = function(vertices, normal, textCoords) {

        gl.bindBuffer(gl.ARRAY_BUFFER, this.iVertexBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STREAM_DRAW);

        gl.bindBuffer(gl.ARRAY_BUFFER, this.iNormalBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(normal), gl.STREAM_DRAW);

        gl.bindBuffer(gl.ARRAY_BUFFER, this.iTextCoordBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textCoords),
gl.STREAM_DRAW);

        this.count = vertices.length/3;
    }

    this.Draw = function() {

        gl.bindBuffer(gl.ARRAY_BUFFER, this.iVertexBuffer);
        gl.vertexAttribPointer(shProgram.iAttribVertex, 3, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(shProgram.iAttribVertex);

        gl.bindBuffer(gl.ARRAY_BUFFER, this.iNormalBuffer);
        gl.vertexAttribPointer(shProgram.iAttribNormal, 3, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(shProgram.iAttribNormal);

        gl.bindBuffer(gl.ARRAY_BUFFER, this.iTextCoordBuffer);
        gl.vertexAttribPointer(shProgram.iAttribTextCoord, 2, gl.FLOAT, false, 0,
0);
        gl.enableVertexAttribArray(shProgram.iAttribTextCoord);

        gl.drawArrays(gl.TRIANGLE_STRIP, 0, this.count);
    }
}

function CreateSurfaceData() {
    let vertexList = [];
    let normalList = [];
    let textCoordList = [];
    let step = 0.01;
    let delta = 0.001;

    for (let r = 0.25; r <= maxR; r += step) {
```

```
        for (let theta = 0; theta < 2 * Math.PI; theta += step) {

            let v1 = equations(r, theta);
            let v2 = equations(r, theta + step);
            let v3 = equations(r + step, theta);
            let v4 = equations(r + step, theta + step);

            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);

            let n1 = CalculateNormal(r, theta, delta);
            let n2 = CalculateNormal(r, theta + step, delta);
            let n3 = CalculateNormal(r + step, theta, delta);
            let n4 = CalculateNormal(r + step, theta + step, delta)

            normalList.push(n1.x, n1.y, n1.z);
            normalList.push(n2.x, n2.y, n2.z);
            normalList.push(n3.x, n3.y, n3.z);

            normalList.push(n2.x, n2.y, n2.z);
            normalList.push(n4.x, n4.y, n4.z);
            normalList.push(n3.x, n3.y, n3.z);

            let t1 = CalculateTextCoord(r, theta);
            let t2 = CalculateTextCoord(r, theta + step);
            let t3 = CalculateTextCoord(r + step, theta);
            let t4 = CalculateTextCoord(r + step, theta + step);

            textCoordList.push(t1.r, t1.theta);
            textCoordList.push(t2.r, t2.theta);
            textCoordList.push(t3.r, t3.theta);

            textCoordList.push(t2.r, t2.theta);
            textCoordList.push(t4.r, t4.theta);
            textCoordList.push(t3.r, t3.theta);
        }
    }

    return { vertices: vertexList, normal: normalList, textCoords: textCoordList };
}

function CalculateTextCoord(r, theta) {

    r = (r - 0.25)/(maxR - 0.25);
    theta = theta / 2*Math.PI;

    return {r, theta};
}
```

```
function LoadTexture() {

    var texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);

    var image = new Image();
    image.crossOrigin = "anonymous";
    image.src = "https://i.ibb.co/1TgPH2f/texture-1.jpg";
    image.addEventListener('load', () => {
        gl.bindTexture(gl.TEXTURE_2D, texture);
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);

        draw();
    }
    );
}

function initGL() {
    let prog = createProgram( gl, vertexShaderSource, fragmentShaderSource );

    shProgram = new ShaderProgram('Basic', prog);
    shProgram.Use();

    shProgram.iAttribVertex              = gl.getAttribLocation(prog, "vertex");
    shProgram.iAttribNormal              = gl.getAttribLocation(prog, "normal");
    shProgram.iModelViewProjectionMatrix = gl.getUniformLocation(prog,
"ModelViewProjectionMatrix");
    shProgram.iModelMatrixNormal         = gl.getUniformLocation(prog,
"ModelNormalMatrix");
    shProgram.iLightPosition             = gl.getUniformLocation(prog,
"lightPosition");
    shProgram.iAttribTextCoord           = gl.getAttribLocation(prog, "textCoord");
    shProgram.iTMU                       = gl.getUniformLocation(prog, "tmu");
    shProgram.iAngleInRadians            = gl.getUniformLocation(prog,
"angleInRadians");
    shProgram.iUserPoint                 = gl.getUniformLocation(prog,
"userPoint");

    surface = new Model('Surface');
    let data = CreateSurfaceData();
    surface.BufferData(data.vertices, data.normal, data.textCoords);

    LoadTexture();

    gl.enable(gl.DEPTH_TEST);
}

function handleKeyPress(event) {
    let stepSize = 0.05;

    switch (event.key) {
```

```
        case 'w':
        case 'W':
            userPoint[0] += stepSize;
            if (userPoint[0] > maxR)
            {
                userPoint[0] = 0.25;
            }
            break;
        case 's':
        case 'S':
            userPoint[0] -= stepSize;
            if (userPoint[0] < 0.25)
            {
                userPoint[0] = maxR;
            }
            break;
        case 'a':
        case 'A':
            userPoint[1] -= stepSize;
            if (userPoint[1] < 0)
            {
                userPoint[1] = 2 * Math.PI;;
            }
            break;
        case 'd':
        case 'D':
            userPoint[1] += stepSize;
            if (userPoint[1] > 2 * Math.PI)
            {
                userPoint[1] = 0;
            }
            break;
        default:
            return;
    }
    updateSurface();
}
```