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

ЗВІТ

ПРО ЛАБОРАТОРНУ РОБОТУ №2

ТЕМА: « РОБОТА З ПРИСТРОЯМИ ВВЕДЕННЯ ТА ОРГАНІЗАЦІЯ ВЗАЄМОДІЇ З КОРИСТУВАЧЕМ »

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

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

Київ 2024

Лабораторна робота №2

РОБОТА З ПРИСТРОЯМИ ВВЕДЕННЯ ТА ОРГАНІЗАЦІЯ ВЗАЄМОДІЇ З КОРИСТУВАЧЕМ

Мета: отримати практичні навички щодо створення та взаємодії з об’єктами за допомогою пристроїв введення.

Завдання:

1. Створити комп’ютерну програму, що містить обробник подій, який створює точки за натисканням на комп’ютерну мишу. Врахувати, що точки зсуваються від курсора миші, що є небажаним. Відповідно, потрібно створити обмежувальний прямокутник канвас в клієнтській області за допомогою event.target.getBoundingClientRect() і виправити позицію курсора, використовуючи ліву та верхню координати цього прямокутника.
2. Додати елемент управління «Button», яка очищає канвас. Створити меню вибору кольору, що буде використовуватися після очищення канвас. Додати меню вибору кольору, щоб встановити колір створюваних точок, що вимагає оновлення шейдерних програм.
3. Забезпечити роботу двох режимів рисування. Перший режим має рисувати точки, другий — будувати трикутник. Додати елемент управління «Button» для кожного з режимів. Під час візуалізації створити вершини в масиві індексів точок як точок і нарисувати вершини в масиві індексів трикутника як трикутники. Намагатися викликати gl.drawArrays якомога менше разів.
4. Додати кнопку для режиму рисування кола. Створити масив для індексів кола. При рисуванні в режимі кола програма має додавати точку при першому натисканні на комп’ютерну мишу, відповідно при другому натисканні додавати вершини для окружності, використовуючи положення, задане під час першого натискання на комп’ютерну мишу, щоб встановити радіус кола. Додати індекс центральної вершини до масиву індексів кіл та видалити цей індекс

із масиву точкових індексів. Нарисувати кожне коло з використанням режиму gl.TRIANGLE_FAN для візуалізації.

1. Створення точок у канвасі

Створюємо блок **canvas**. Створюємо обробник подій для кліку миші, який додає точки на канвас за натисканням. Використовуємо метод **event.target.getBoundingClientRect()**, щоб отримати координати прямокутника канвас і правильно розрахувати позицію кліку. Це допоможе уникнути зсуву точки від курсора миші. (рис. 1.1).

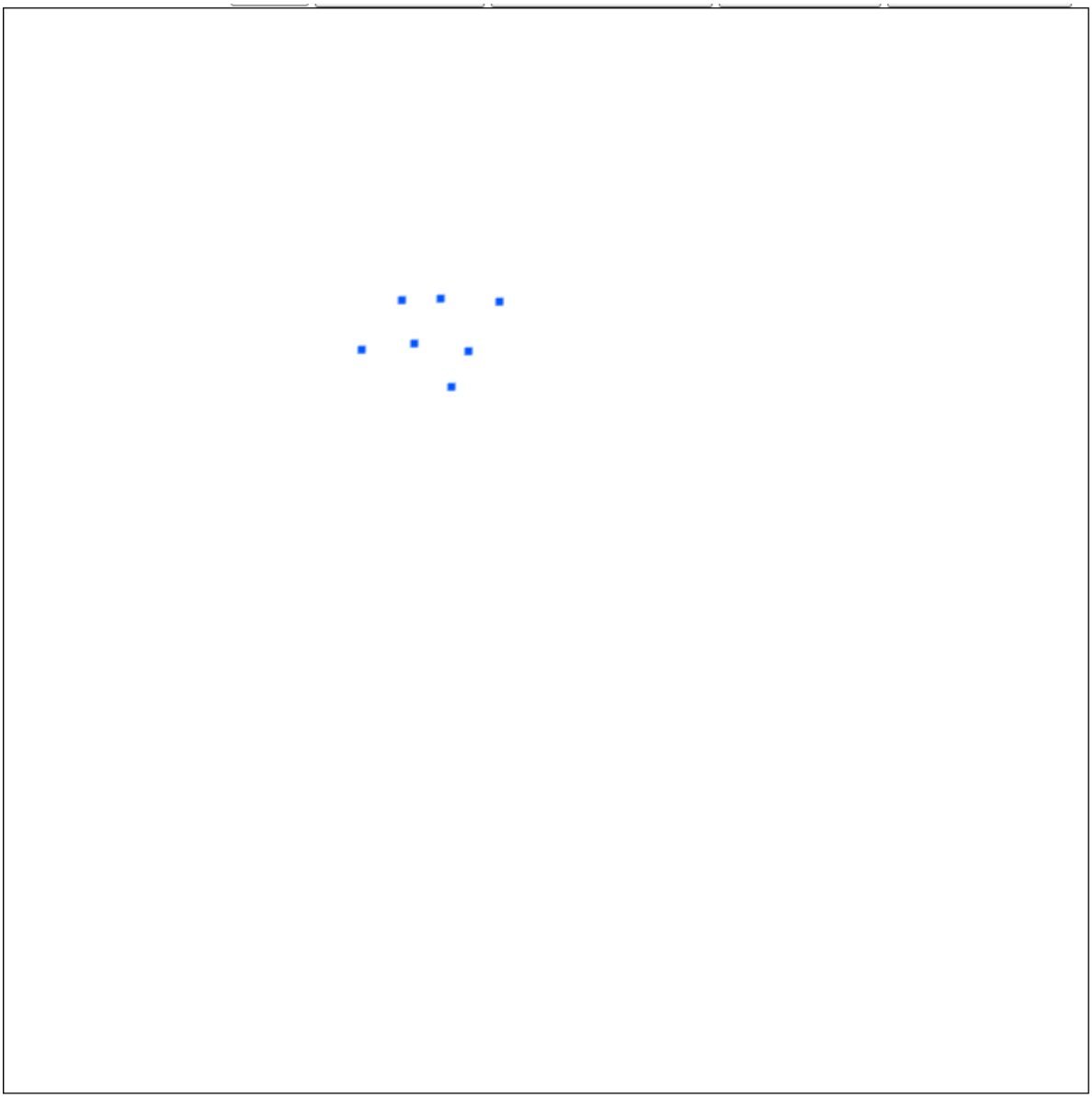

рис. 1.1. Демонстрація виконання першого пункту завдання

2. Очищення канвасу та вибір кольору

Додаємо елемент керування **Button**, який очищає канвас. Додаємо меню вибору кольору, яке задає колір створюваних точок. Меню вибору кольору оновлює шейдерну програму для відображення кольорів. (рис. 2.1).

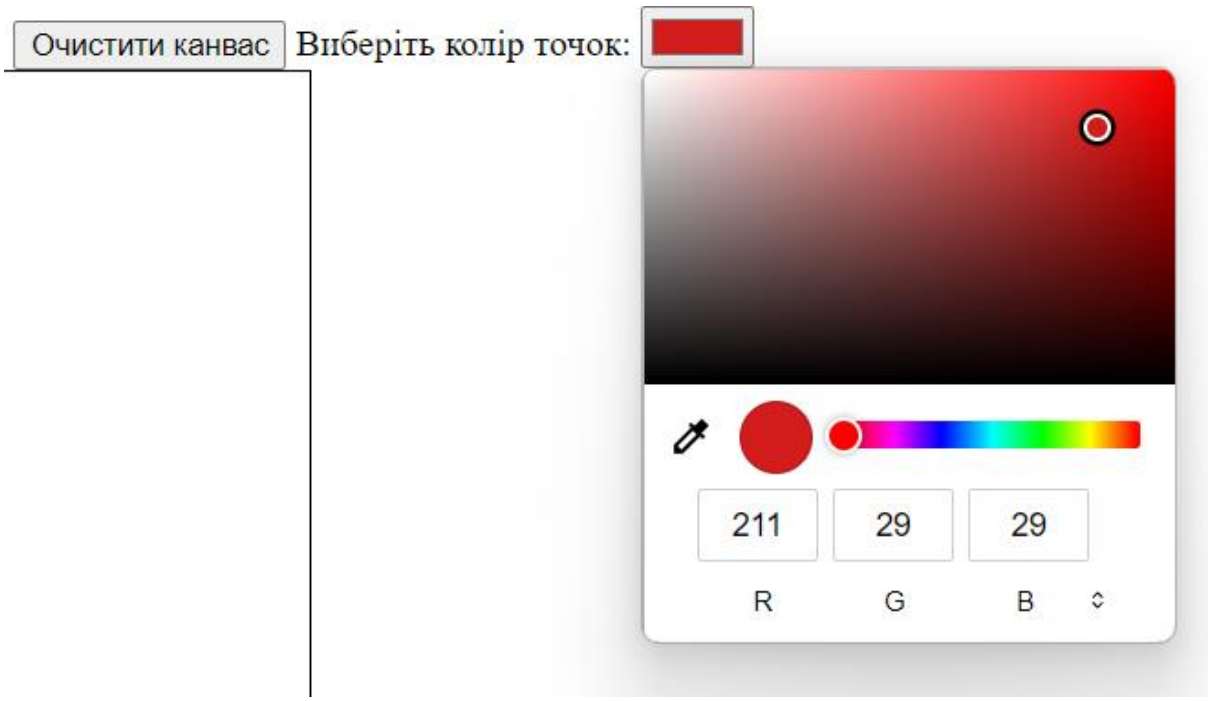

рис. 2.1. Результат виконання другого пункту завдання

3. Створення декількох режимів малювання

Створюємо два режими рисування. Перший режим – для рисування точок, другий – для побудови трикутників. Додаємо кнопки для перемикання між режимами. У режимі точок створюємо масив індексів для точок, у режимі трикутників – для трикутників. Викликаємо функцію **gl.drawArrays** якомога рідше, щоб підвищити продуктивність. (рис. 3.1).

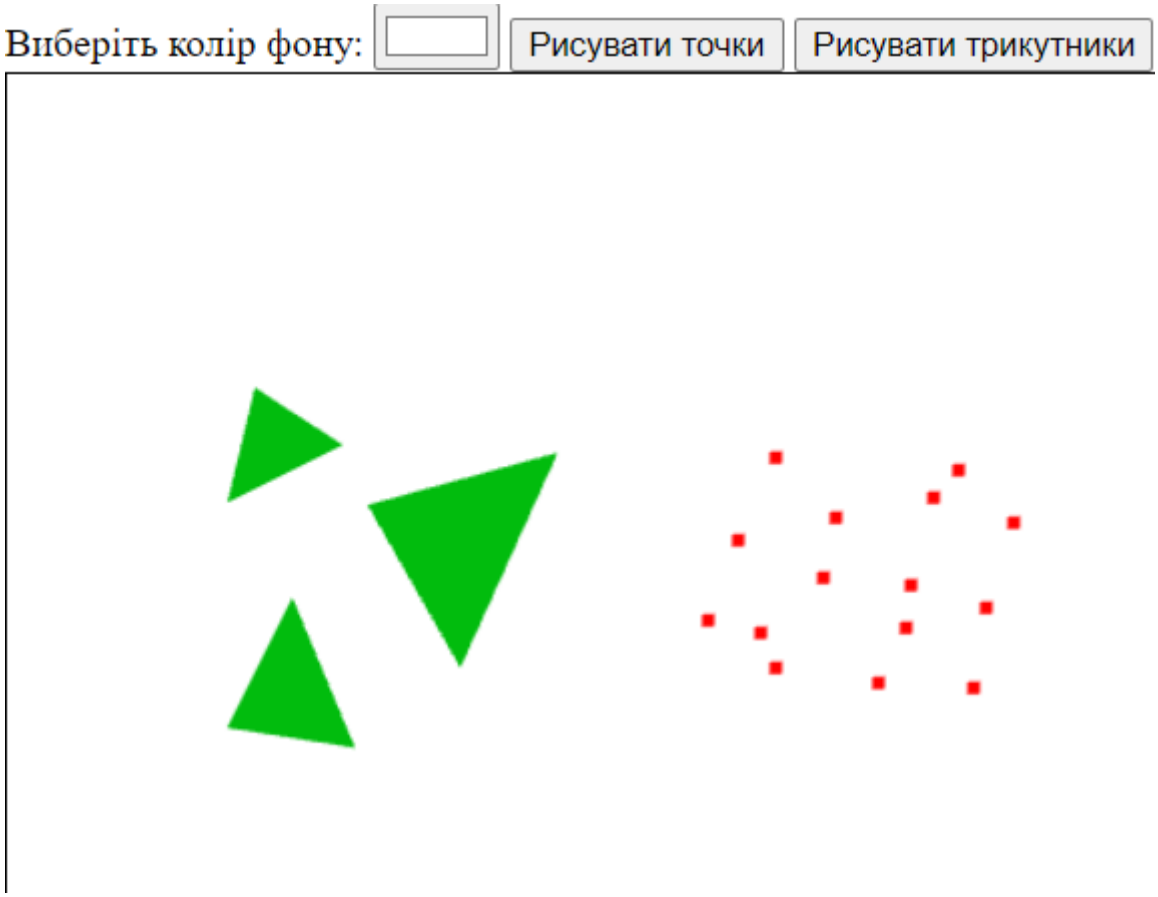

рис. 3.1. Результат виконання третього пункту завдання

4. Режим малювання кола

Додаємо кнопку для режиму рисування кіл. Створюємо масив індексів для кіл. У режимі кола перше натискання на мишу додає центральну точку кола, а друге натискання встановлює радіус і додає вершини для окружності. Індекси центральної точки додаються до масиву індексів кола, а з масиву індексів точок ця точка видаляється. Для візуалізації кола використовуємо режим **gl.TRIANGLE_FAN**. (рис. 4.1)

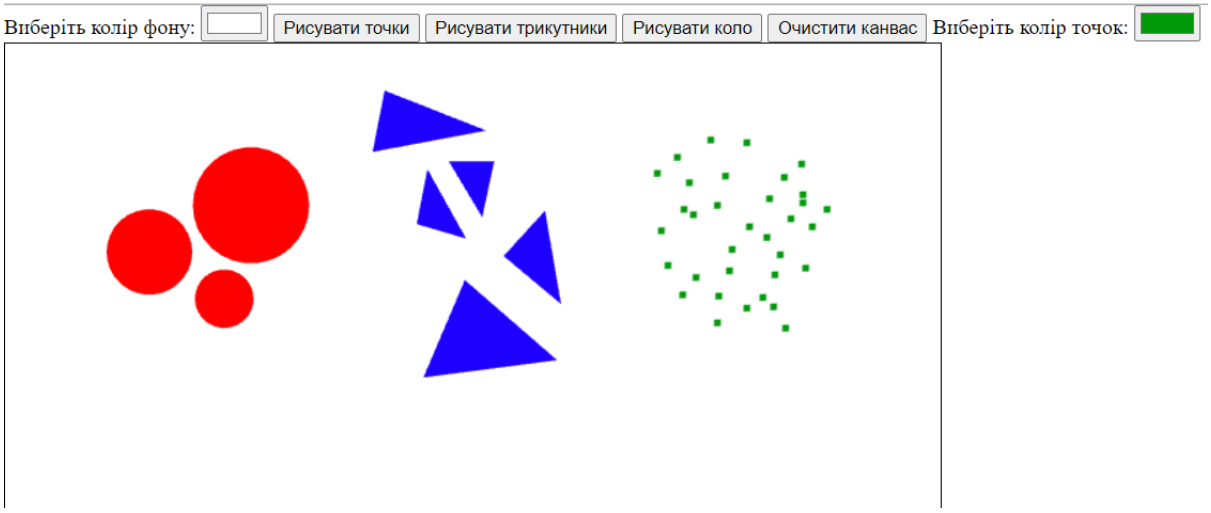

рис. 4.1. Результат виконання четвертого пункту завдання

Висновок: Під час виконання даної лабораторної роботи ми навчилися працювати з подіями миші для побудови графічних об'єктів на канвасі, правильно визначати позицію курсора за допомогою методу getBoundingClientRect(), а також реалізовувати кілька режимів рисування: точки, трикутники та кола. Ми додали елементи керування для очищення канваса та зміни кольору об'єктів, а також оптимізували виклики gl.drawArrays для підвищення продуктивності.

ПРОГРАМНИЙ КОД

HTML:

```
<!DOCTYPE HTML>
<html>
   <head>
      <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Lab 2 / User interaction</title>
      <style>
         body, html { margin: 0; padding: 0; overflow: hidden; }
         canvas { display: block; border: 1px solid black;}
      </style>
   </head>

   <body>
      <label for="bgColorPicker">Виберіть колір фону:</label>
      <input type="color" id="bgColorPicker" value="#e6e6e6">
      <button id="drawPointsBtn">Рисувати точки</button>
      <button id="drawTrianglesBtn">Рисувати трикутники</button>
      <button id="drawCircleBtn">Рисувати коло</button>
      <button id="clearBtn">Очистити канвас</button>

      <label for="colorPicker">Виберіть колір точок:</label>
      <input type="color" id="colorPicker" name="colorPicker"
value="#ff0000">

      <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/user_interaction.js"></script>
   </body>
</html>
```

JS:

```
function setupUserInteraction() {
```

```
const canvas = document.getElementById('mycanvas');
const gl = canvas.getContext('webgl');

if (!gl) {
    alert("Ваш браузер не підтримує WebGL");
    return;
}

let drawMode = 'points';
let points = [];
let triangles = [];
let circles = [];
let selectedColor = [1.0, 0.0, 0.0, 1.0];
let firstClick = false;
let circleCenter = null;
const circleVerticesCount = 30;

const clearBtn = document.getElementById('clearBtn');
const drawPointsBtn = document.getElementById('drawPointsBtn');
const drawTrianglesBtn = document.getElementById('drawTrianglesBtn');
const drawCircleBtn = document.getElementById('drawCircleBtn');
const colorPicker = document.getElementById('colorPicker');
const bgColorPicker = document.getElementById('bgColorPicker');

const vertexShaderSource = `
    attribute vec4 aPosition;
    attribute vec4 aColor;
    varying lowp vec4 vColor;
    void main(void) {
        gl_Position = aPosition;
        gl_PointSize = 5.0;
        vColor = aColor;
    }
`;

const fragmentShaderSource = `
    precision mediump float;
```

```
    varying lowp vec4 vColor;
    void main(void) {
        gl_FragColor = vColor;
    }
`;

function createShader(gl, type, source) {
    const shader = gl.createShader(type);
    gl.shaderSource(shader, source);
    gl.compileShader(shader);
    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        console.error('Помилка компіляції шейдера',
gl.getShaderInfoLog(shader));
        gl.deleteShader(shader);
        return null;
    }
    return shader;
}

function createProgram(gl, vertexShader, fragmentShader) {
    const program = gl.createProgram();
    gl.attachShader(program, vertexShader);
    gl.attachShader(program, fragmentShader);
    gl.linkProgram(program);
    if (!gl.getProgramParameter(program, gl.LINK_STATUS)) {
        console.error('Помилка лінкування програми',
gl.getProgramInfoLog(program));
        gl.deleteProgram(program);
        return null;
    }
    return program;
}

const vertexShader = createShader(gl, gl.VERTEX_SHADER,
vertexShaderSource);
const fragmentShader = createShader(gl, gl.FRAGMENT_SHADER,
fragmentShaderSource);
```

```
const program = createProgram(gl, vertexShader, fragmentShader);
gl.useProgram(program);

const aPosition = gl.getAttribLocation(program, 'aPosition');
const aColor = gl.getAttribLocation(program, 'aColor');
const positionBuffer = gl.createBuffer();
const colorBuffer = gl.createBuffer();

colorPicker.addEventListener('input', (event) => {
    const hexColor = event.target.value;
    selectedColor = hexToRgbA(hexColor);
});

bgColorPicker.addEventListener('input', (event) => {
    const hexBgColor = event.target.value;
    const [r, g, b] = hexToRgb(hexBgColor);
    gl.clearColor(r, g, b, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);
});

function hexToRgbA(hex) {
    let r = parseInt(hex.slice(1, 3), 16) / 255;
    let g = parseInt(hex.slice(3, 5), 16) / 255;
    let b = parseInt(hex.slice(5, 7), 16) / 255;
    return [r, g, b, 1.0];
}

function hexToRgb(hex) {
    let r = parseInt(hex.slice(1, 3), 16) / 255;
    let g = parseInt(hex.slice(3, 5), 16) / 255;
    let b = parseInt(hex.slice(5, 7), 16) / 255;
    return [r, g, b];
}

canvas.addEventListener('mousedown', (event) => {
    const rect = event.target.getBoundingClientRect();
    const x = ((event.clientX - rect.left) / canvas.width) * 2 - 1;
```

```
    const y = ((event.clientY - rect.top) / canvas.height) * -2 + 1;

    if (drawMode === 'points') {
        points.push(x, y, ...selectedColor);
    } else if (drawMode === 'triangles') {
        points.push(x, y, ...selectedColor);
        if (points.length >= 18) {
            triangles.push(...points.splice(0, 18));
        }
    } else if (drawMode === 'circle') {
        if (!firstClick) {
            circleCenter = [x, y];
            firstClick = true;
        } else {
            const radius = Math.sqrt(Math.pow(x - circleCenter[0], 2) +
Math.pow(y - circleCenter[1], 2));
            let newCircle = [circleCenter[0], circleCenter[1], ...selectedColor];
            for (let i = 0; i <= circleVerticesCount; i++) {
                const angle = (i / circleVerticesCount) * Math.PI * 2;
                const circleX = circleCenter[0] + Math.cos(angle) * radius;
                const circleY = circleCenter[1] + Math.sin(angle) * radius;
                newCircle.push(circleX, circleY, ...selectedColor);
            }
            circles.push(newCircle);
            firstClick = false;
        }
    }

    drawScene();
});

drawPointsBtn.addEventListener('click', () => {
    drawMode = 'points';
    firstClick = false;
});

drawTrianglesBtn.addEventListener('click', () => {
```

```
    drawMode = 'triangles';
    firstClick = false;
  });

  drawCircleBtn.addEventListener('click', () => {
    drawMode = 'circle';
    firstClick = false;
  });

  clearBtn.addEventListener('click', () => {
    points = [];
    triangles = [];
    circles = [];
    gl.clear(gl.COLOR_BUFFER_BIT);
  });

  function drawScene() {
    gl.clear(gl.COLOR_BUFFER_BIT);

    if (points.length > 0) {
      gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
      gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(points),
gl.STATIC_DRAW);
      gl.vertexAttribPointer(aPosition, 2, gl.FLOAT, false, 24, 0);
      gl.enableVertexAttribArray(aPosition);

      gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
      gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(points),
gl.STATIC_DRAW);
      gl.vertexAttribPointer(aColor, 4, gl.FLOAT, false, 24, 8);
      gl.enableVertexAttribArray(aColor);

      gl.drawArrays(gl.POINTS, 0, points.length / 6);
    }

    if (triangles.length > 0) {
      gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
```

```
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangles),
gl.STATIC_DRAW);
        gl.vertexAttribPointer(aPosition, 2, gl.FLOAT, false, 24, 0);
        gl.enableVertexAttribArray(aPosition);

        gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangles),
gl.STATIC_DRAW);
        gl.vertexAttribPointer(aColor, 4, gl.FLOAT, false, 24, 8);
        gl.enableVertexAttribArray(aColor);

        gl.drawArrays(gl.TRIANGLES, 0, triangles.length / 6);
    }

    if (circles.length > 0) {
        circles.forEach(circle => {
            gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
            gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(circle),
gl.STATIC_DRAW);
            gl.vertexAttribPointer(aPosition, 2, gl.FLOAT, false, 24, 0);
            gl.enableVertexAttribArray(aPosition);

            gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
            gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(circle),
gl.STATIC_DRAW);
            gl.vertexAttribPointer(aColor, 4, gl.FLOAT, false, 24, 8);
            gl.enableVertexAttribArray(aColor);

            const numVertices = circleVerticesCount + 2;
            gl.drawArrays(gl.TRIANGLE_FAN, 0, numVertices);
        });
    }
  }

  gl.clearColor(0.9, 0.9, 0.9, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT);
}
```

```
window.onload = setupUserInteraction;
```