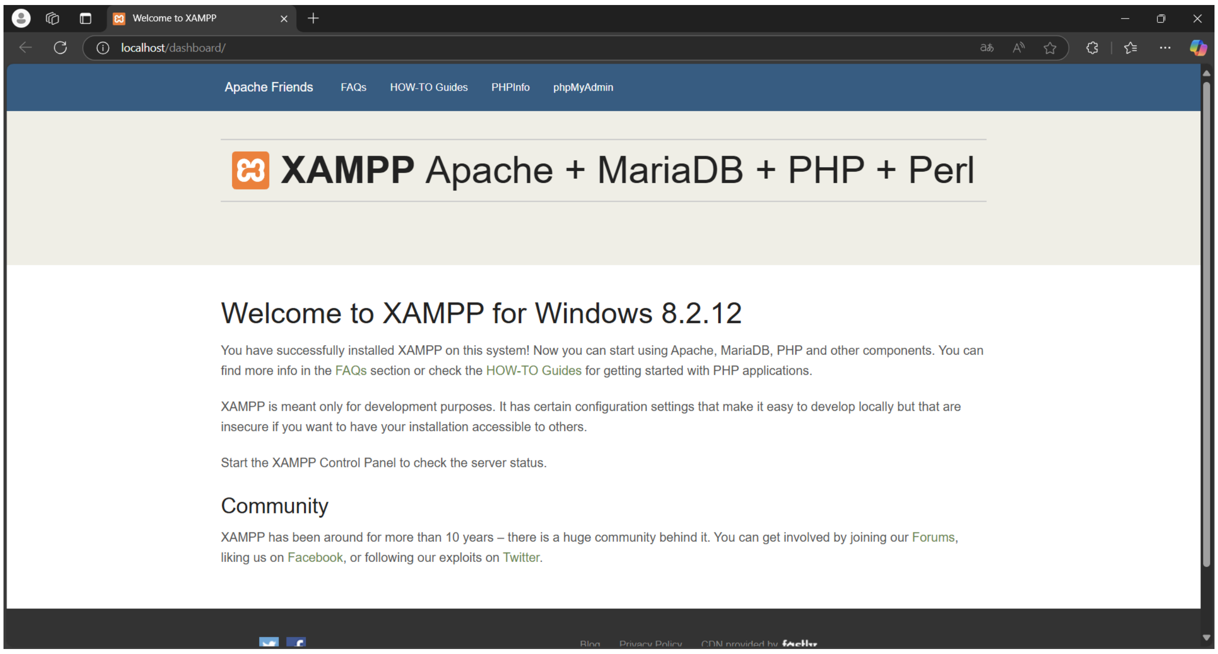
Part 1

Here is the working web server in Figure 1 with the help of the XAMPP application.



XAMPP Apache + MariaDB + PHP + Perl

Welcome to XAMPP for Windows 8.2.12

You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the FAQs section or check the HOW-TO Guides for getting started with PHP applications.

XAMPP is meant only for development purposes. It has certain configuration settings that make it easy to develop locally but that are insecure if you want to have your installation accessible to others.

Start the XAMPP Control Panel to check the server status.

Community

XAMPP has been around for more than 10 years – there is a huge community behind it. You can get involved by joining our Forums, liking us on Facebook, or following our exploits on Twitter.

Part 2

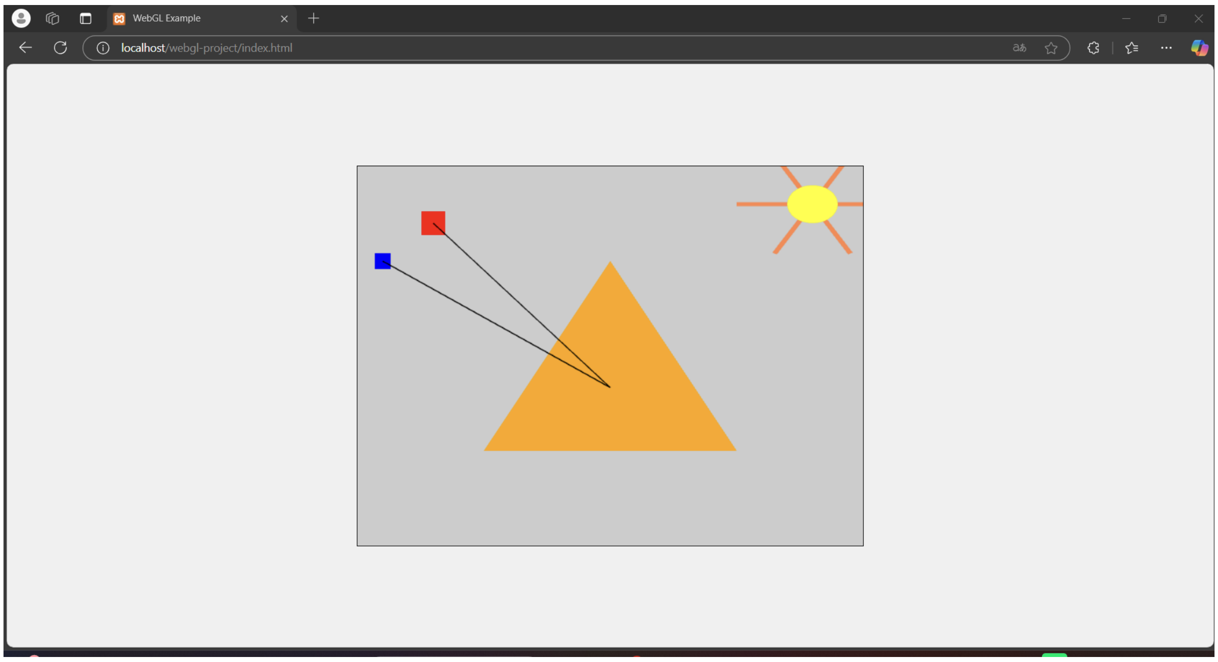
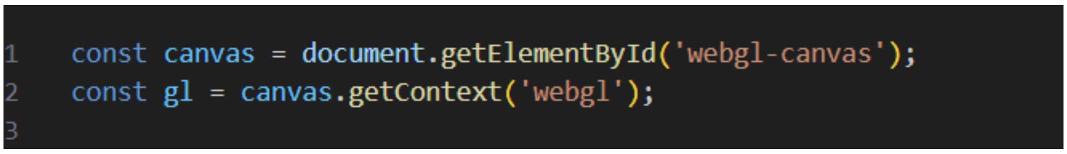


Figure 2: My result as a full page with pyramid and the sun at the right upper corner. Also, you can see the orange lines coming out from the sun as light. And UFOs (points) of different sizes attacking the pyramid with black lines at the left upper corner.

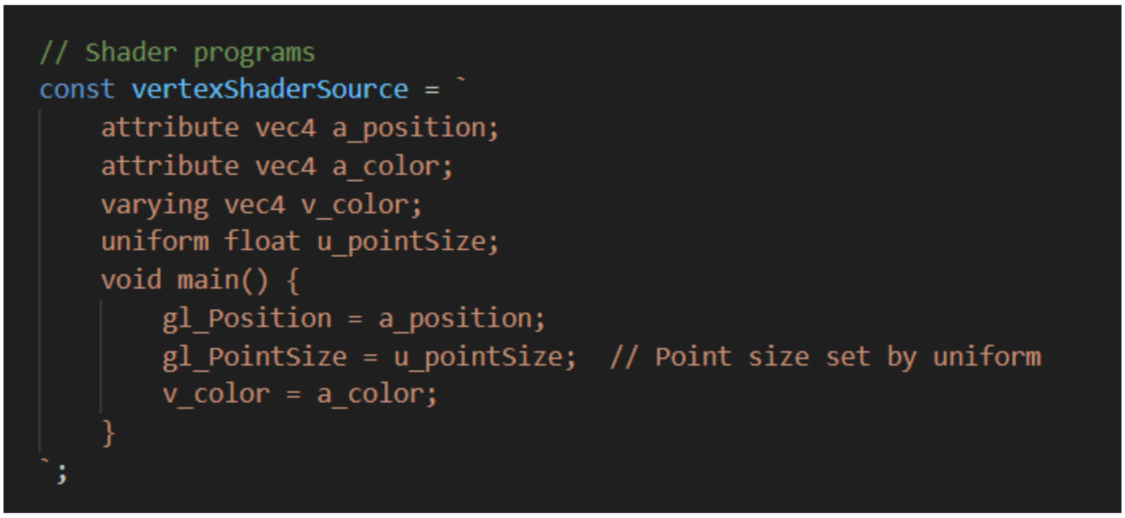
1. **WebGL Context Setup**



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

```

- This section initializes the WebGL context (gl) for drawing on the canvas element.

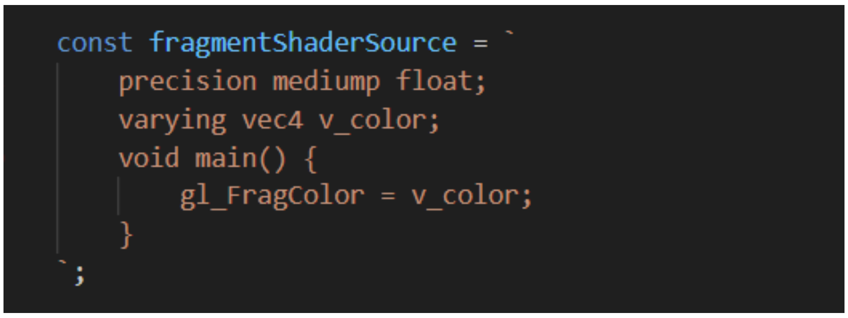
2. **Shader Program Definition**



```javascript
// Shader programs
const vertexShaderSource = `
    attribute vec4 a_position;
    attribute vec4 a_color;
    varying vec4 v_color;
    uniform float u_pointSize;
    void main() {
        gl_Position = a_position;
        gl_PointSize = u_pointSize;  // Point size set by uniform
        v_color = a_color;
    }
`;
```

- Takes in position (a_position) and color (a_color) attributes for each vertex.
- Sets the position and size (u_pointSize for points) of each vertex, and passes the color to the fragment shader.

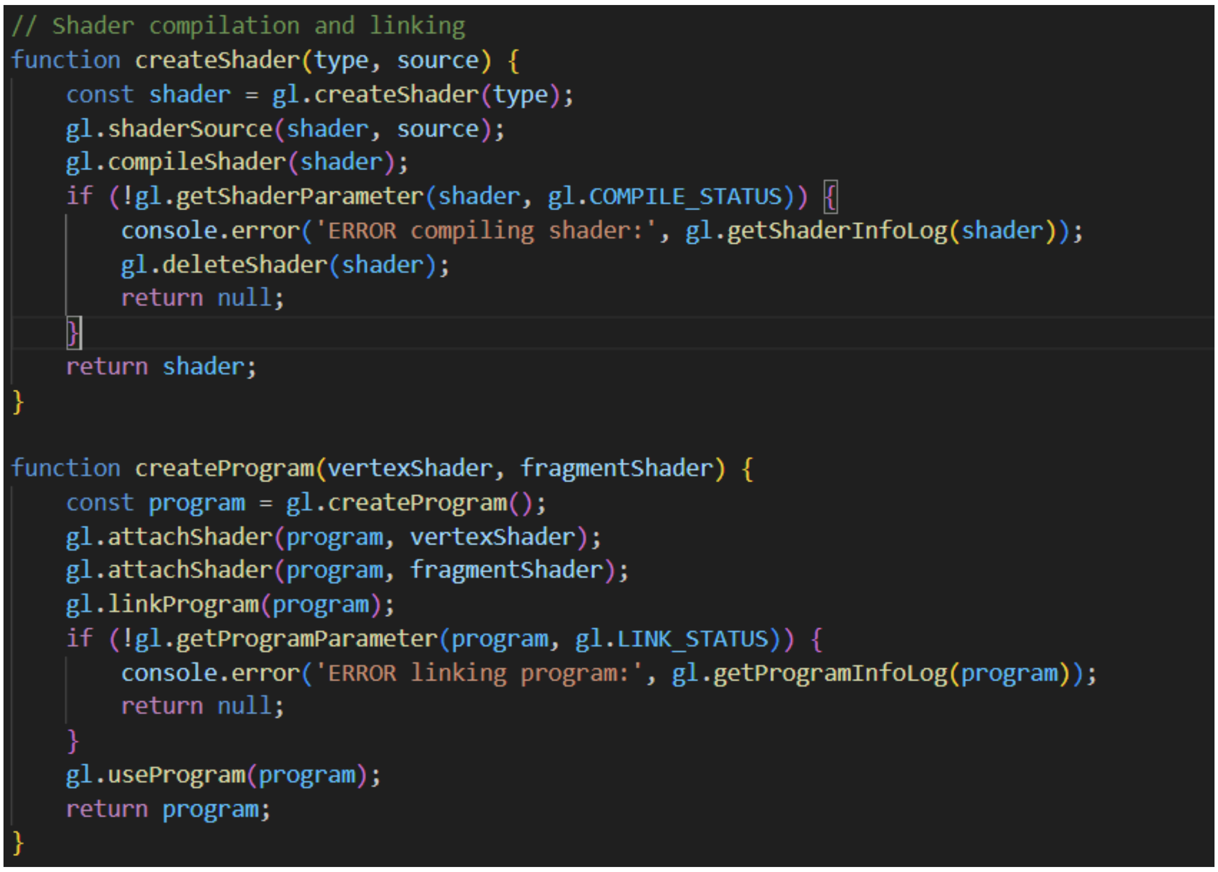
Fragment Shader (fragmentShaderSource):



```javascript
const fragmentShaderSource = `
    precision mediump float;
    varying vec4 v_color;
    void main() {
        gl_FragColor = v_color;
    }
`;
```

- This shader uses the interpolated color (v_color) from the vertex shader to color each pixel.

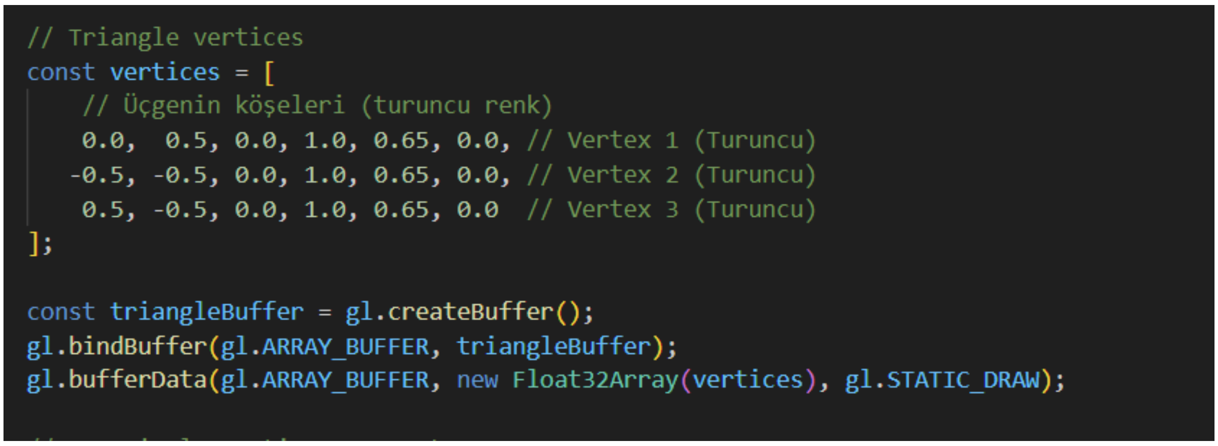
3. **Shader Compilation and Program Linking**



```javascript
// Shader compilation and linking
function createShader(type, source) {
    const shader = gl.createShader(type);
    gl.shaderSource(shader, source);
    gl.compileShader(shader);
    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        console.error('ERROR compiling shader:', gl.getShaderInfoLog(shader));
        gl.deleteShader(shader);
        return null;
    }
    return shader;
}

function createProgram(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('ERROR linking program:', gl.getProgramInfoLog(program));
        return null;
    }
    gl.useProgram(program);
    return program;
}
```

- createShader: Compiles a shader of a given type (gl.VERTEX_SHADER or gl.FRAGMENT_SHADER) and logs any compilation errors.
- createProgram: Links the vertex and fragment shaders into a WebGL program and logs linking errors.

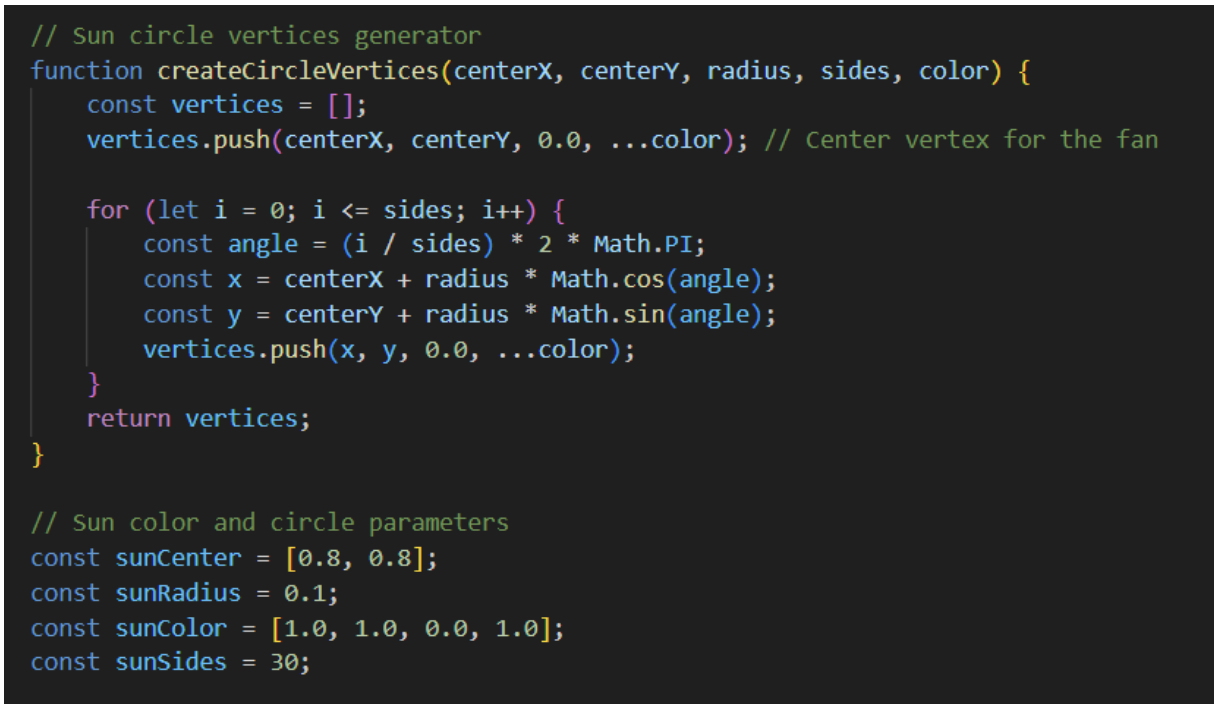
4. **Triangle Definition and Buffer Setup**



```javascript
// Triangle vertices
const vertices = [
    // Üçgenin köşeleri (turuncu renk)
    0.0,  0.5, 0.0, 1.0, 0.65, 0.0, // Vertex 1 (Turuncu)
   -0.5, -0.5, 0.0, 1.0, 0.65, 0.0, // Vertex 2 (Turuncu)
    0.5, -0.5, 0.0, 1.0, 0.65, 0.0  // Vertex 3 (Turuncu)
];

const triangleBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, triangleBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
```

- Defines the triangle’s vertices, with orange color values for each vertex.
- Binds the triangle data to a buffer (triangleBuffer) for rendering.

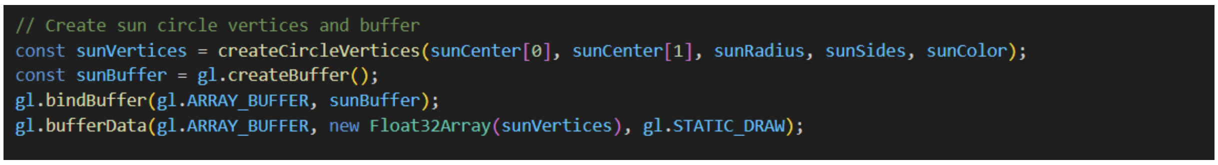
5. **Sun Circle Creation**



```
// Sun circle vertices generator
function createCircleVertices(centerX, centerY, radius, sides, color) {
    const vertices = [];
    vertices.push(centerX, centerY, 0.0, ...color); // Center vertex for the fan

    for (let i = 0; i <= sides; i++) {
        const angle = (i / sides) * 2 * Math.PI;
        const x = centerX + radius * Math.cos(angle);
        const y = centerY + radius * Math.sin(angle);
        vertices.push(x, y, 0.0, ...color);
    }
    return vertices;
}

// Sun color and circle parameters
const sunCenter = [0.8, 0.8];
const sunRadius = 0.1;
const sunColor = [1.0, 1.0, 0.0, 1.0];
const sunSides = 30;
```

- Generates vertices for a circle using a triangle fan pattern, centered at (centerX, centerY) with a specified radius and color(yellow).

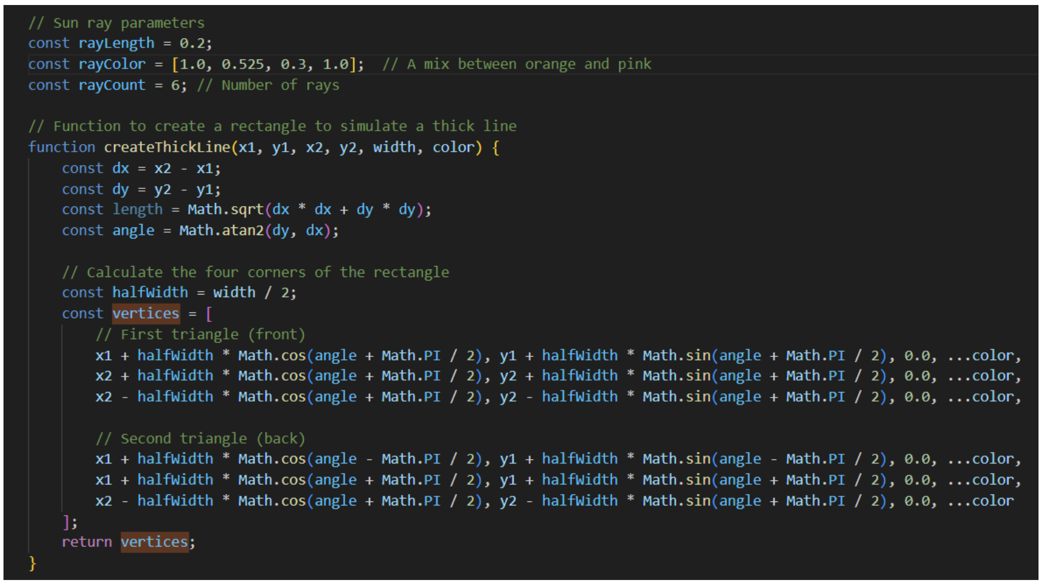
Sun Circle Parameters and Buffer Setup:



```
// Create sun circle vertices and buffer
const sunVertices = createCircleVertices(sunCenter[0], sunCenter[1], sunRadius, sunSides, sunColor);
const sunBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, sunBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(sunVertices), gl.STATIC_DRAW);
```

- Creates the sun vertices using createCircleVertices, then binds them to a buffer (sunBuffer) for rendering.

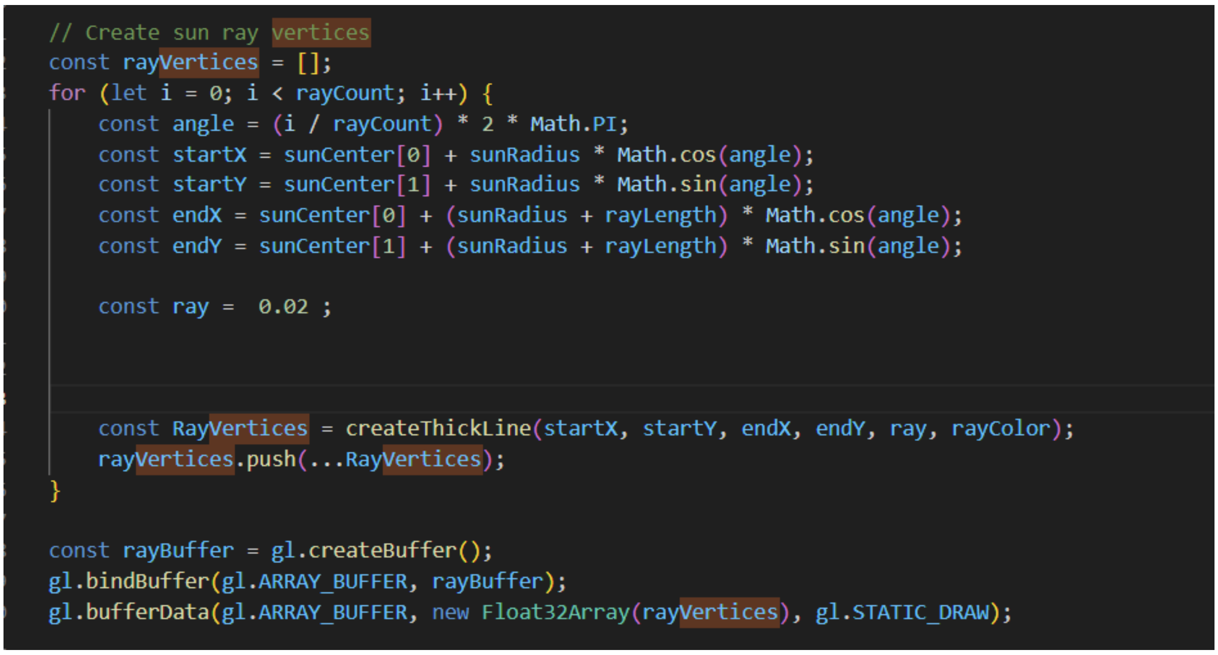
6. **Sun Rays Creation**



```
// Sun ray parameters
const rayLength = 0.2;
const rayColor = [1.0, 0.525, 0.3, 1.0];  // A mix between orange and pink
const rayCount = 6; // Number of rays

// Function to create a rectangle to simulate a thick line
function createThickLine(x1, y1, x2, y2, width, color) {
    const dx = x2 - x1;
    const dy = y2 - y1;
    const length = Math.sqrt(dx * dx + dy * dy);
    const angle = Math.atan2(dy, dx);

    // Calculate the four corners of the rectangle
    const halfWidth = width / 2;
    const vertices = [
        // First triangle (front)
        x1 + halfWidth * Math.cos(angle + Math.PI / 2), y1 + halfWidth * Math.sin(angle + Math.PI / 2), 0.0, ...color,
        x2 + halfWidth * Math.cos(angle + Math.PI / 2), y2 + halfWidth * Math.sin(angle + Math.PI / 2), 0.0, ...color,
        x2 - halfWidth * Math.cos(angle + Math.PI / 2), y2 - halfWidth * Math.sin(angle + Math.PI / 2), 0.0, ...color,

        // Second triangle (back)
        x1 + halfWidth * Math.cos(angle - Math.PI / 2), y1 + halfWidth * Math.sin(angle - Math.PI / 2), 0.0, ...color,
        x1 + halfWidth * Math.cos(angle + Math.PI / 2), y1 + halfWidth * Math.sin(angle + Math.PI / 2), 0.0, ...color,
        x2 - halfWidth * Math.cos(angle + Math.PI / 2), y2 - halfWidth * Math.sin(angle + Math.PI / 2), 0.0, ...color
    ];
    return vertices;
}
```

- Takes two endpoints (x1, y1) and (x2, y2), a line width, and color to generate vertices for a rectangle. Lines are orange-pink colored.

Generating Ray Vertices:

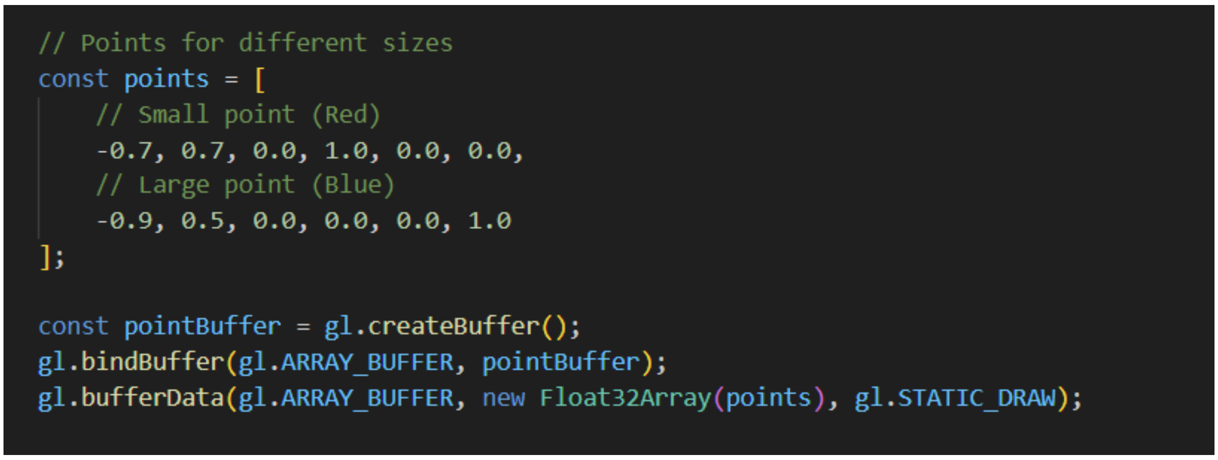


```
// Create sun ray vertices
const rayVertices = [];
for (let i = 0; i < rayCount; i++) {
    const angle = (i / rayCount) * 2 * Math.PI;
    const startX = sunCenter[0] + sunRadius * Math.cos(angle);
    const startY = sunCenter[1] + sunRadius * Math.sin(angle);
    const endX = sunCenter[0] + (sunRadius + rayLength) * Math.cos(angle);
    const endY = sunCenter[1] + (sunRadius + rayLength) * Math.sin(angle);

    const ray =  0.02 ;



    const RayVertices = createThickLine(startX, startY, endX, endY, ray, rayColor);
    rayVertices.push(...RayVertices);
}

const rayBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, rayBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(rayVertices), gl.STATIC_DRAW);
```

- Loops to create each ray at an angle around the circle.

7. **Points with Different Sizes**

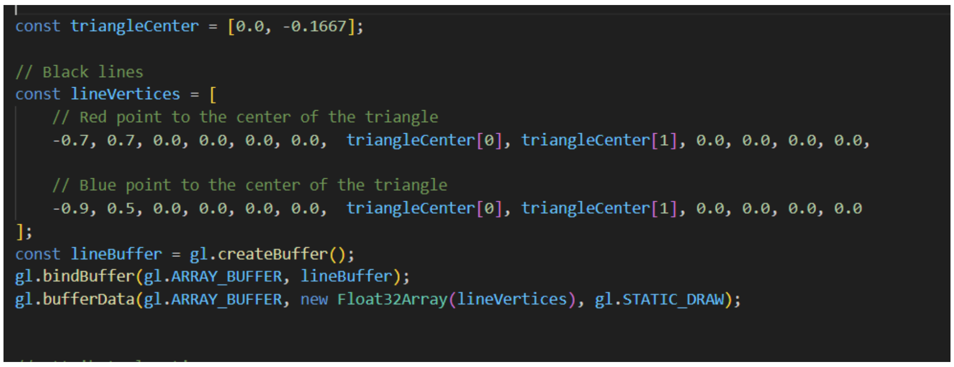
Point Definitions:



```
// Points for different sizes
const points = [
    // Small point (Red)
    -0.7, 0.7, 0.0, 1.0, 0.0, 0.0,
    // Large point (Blue)
    -0.9, 0.5, 0.0, 0.0, 0.0, 1.0
];

const pointBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, pointBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(points), gl.STATIC_DRAW);
```

- Defines two points with blue and red and positions, stored in pointBuffer.

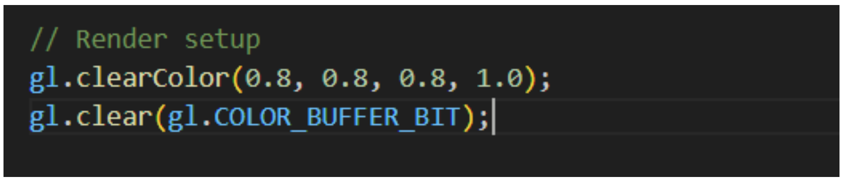
8.Generating Black Lines



```
const triangleCenter = [0.0, -0.1667];

// Black lines
const lineVertices = [
    // Red point to the center of the triangle
    -0.7, 0.7, 0.0, 0.0, 0.0, 0.0,  triangleCenter[0], triangleCenter[1], 0.0, 0.0, 0.0, 0.0,

    // Blue point to the center of the triangle
    -0.9, 0.5, 0.0, 0.0, 0.0, 0.0,  triangleCenter[0], triangleCenter[1], 0.0, 0.0, 0.0, 0.0
];
const lineBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, lineBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(lineVertices), gl.STATIC_DRAW);
```

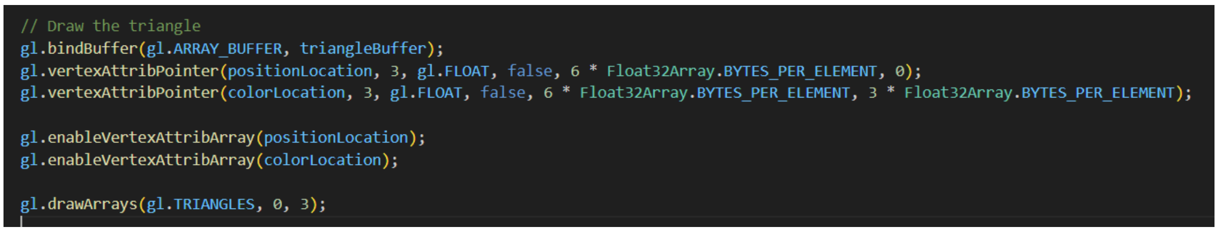
- Black lines are generated as the UFO attack to the center of the pyramid.

8. **Rendering Setup and Draw Calls**



```
// Render setup
gl.clearColor(0.8, 0.8, 0.8, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);
```

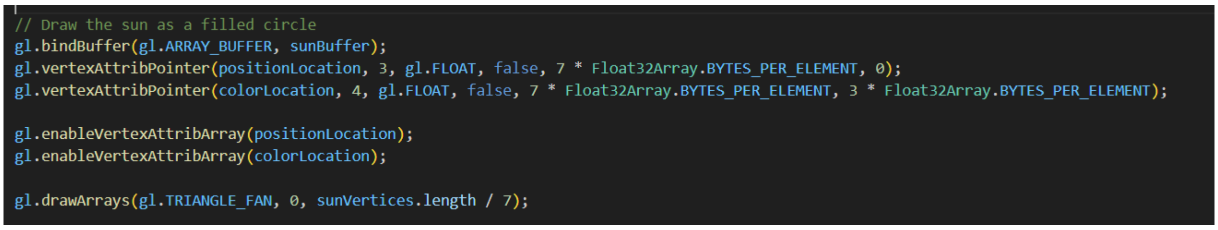
Drawing Triangle:



```
// Draw the triangle
gl.bindBuffer(gl.ARRAY_BUFFER, triangleBuffer);
gl.vertexAttribPointer(positionLocation, 3, gl.FLOAT, false, 6 * Float32Array.BYTES_PER_ELEMENT, 0);
gl.vertexAttribPointer(colorLocation, 3, gl.FLOAT, false, 6 * Float32Array.BYTES_PER_ELEMENT, 3 * Float32Array.BYTES_PER_ELEMENT);

gl.enableVertexAttribArray(positionLocation);
gl.enableVertexAttribArray(colorLocation);

gl.drawArrays(gl.TRIANGLES, 0, 3);
```

- Binds the triangleBuffer, sets up attributes for position and color, and draws the triangle. TRIANGLES is used.

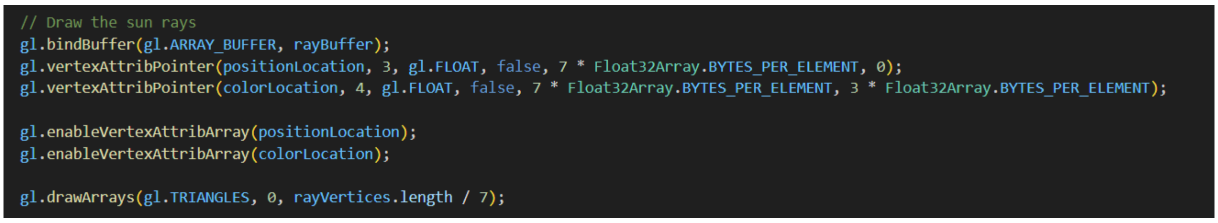
Drawing Sun Circle:



```
// Draw the sun as a filled circle
gl.bindBuffer(gl.ARRAY_BUFFER, sunBuffer);
gl.vertexAttribPointer(positionLocation, 3, gl.FLOAT, false, 7 * Float32Array.BYTES_PER_ELEMENT, 0);
gl.vertexAttribPointer(colorLocation, 4, gl.FLOAT, false, 7 * Float32Array.BYTES_PER_ELEMENT, 3 * Float32Array.BYTES_PER_ELEMENT);

gl.enableVertexAttribArray(positionLocation);
gl.enableVertexAttribArray(colorLocation);

gl.drawArrays(gl.TRIANGLE_FAN, 0, sunVertices.length / 7);
```

- Binds sunBuffer, sets up attributes, and draws the sun as a filled circle. TRIANGLE_FAN is used.

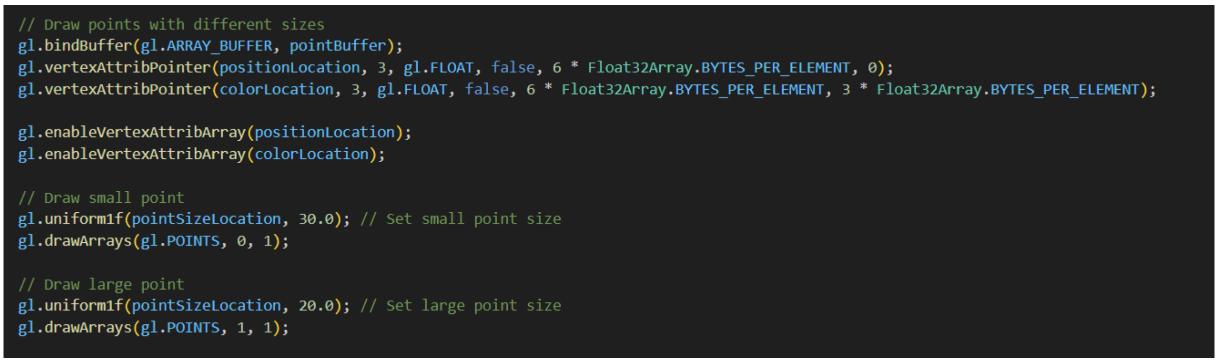
Drawing Sun Rays:



```
// Draw the sun rays
gl.bindBuffer(gl.ARRAY_BUFFER, rayBuffer);
gl.vertexAttribPointer(positionLocation, 3, gl.FLOAT, false, 7 * Float32Array.BYTES_PER_ELEMENT, 0);
gl.vertexAttribPointer(colorLocation, 4, gl.FLOAT, false, 7 * Float32Array.BYTES_PER_ELEMENT, 3 * Float32Array.BYTES_PER_ELEMENT);

gl.enableVertexAttribArray(positionLocation);
gl.enableVertexAttribArray(colorLocation);

gl.drawArrays(gl.TRIANGLES, 0, rayVertices.length / 7);
```

- Binds rayBuffer, sets up attributes, and draws each ray as triangles. TRIANGLES is used.

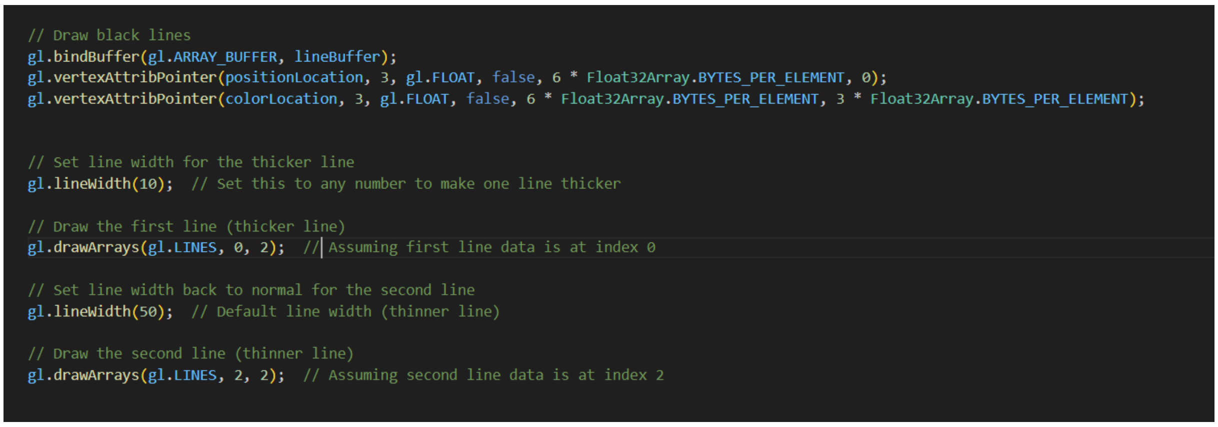
Drawing Points with Varying Sizes:



```
// Draw points with different sizes
gl.bindBuffer(gl.ARRAY_BUFFER, pointBuffer);
gl.vertexAttribPointer(positionLocation, 3, gl.FLOAT, false, 6 * Float32Array.BYTES_PER_ELEMENT, 0);
gl.vertexAttribPointer(colorLocation, 3, gl.FLOAT, false, 6 * Float32Array.BYTES_PER_ELEMENT, 3 * Float32Array.BYTES_PER_ELEMENT);

gl.enableVertexAttribArray(positionLocation);
gl.enableVertexAttribArray(colorLocation);

// Draw small point
gl.uniform1f(pointSizeLocation, 30.0); // Set small point size
gl.drawArrays(gl.POINTS, 0, 1);

// Draw large point
gl.uniform1f(pointSizeLocation, 20.0); // Set large point size
gl.drawArrays(gl.POINTS, 1, 1);
```

- Binds pointBuffer and sets point sizes (u_pointSize) for the small and large points. POINTS is used.

Drawing Lines:



```
// Draw black lines
gl.bindBuffer(gl.ARRAY_BUFFER, lineBuffer);
gl.vertexAttribPointer(positionLocation, 3, gl.FLOAT, false, 6 * Float32Array.BYTES_PER_ELEMENT, 0);
gl.vertexAttribPointer(colorLocation, 3, gl.FLOAT, false, 6 * Float32Array.BYTES_PER_ELEMENT, 3 * Float32Array.BYTES_PER_ELEMENT);


// Set line width for the thicker line
gl.lineWidth(10);  // Set this to any number to make one line thicker

// Draw the first line (thicker line)
gl.drawArrays(gl.LINES, 0, 2);  // Assuming first line data is at index 0

// Set line width back to normal for the second line
gl.lineWidth(50);  // Default line width (thinner line)

// Draw the second line (thinner line)
gl.drawArrays(gl.LINES, 2, 2);  // Assuming second line data is at index 2
```

- Binds linebuffer, sets up the position and color attributes for the lines, and then draws the black lines using the LINES. **The sizes of the lines are different as the requiement of the Project.Unless lineWidth() function is not supported by the browser we used**. Since we are responsible for using LINES this method did not work but if we were allowed to use Narrow Strip Triangle, that could be achieved.

In summary, POINTS, TRIANGLE,TRIANGLE_FAN and LINES are used, at least 3 different colors are added, points with different sizes are generated, lines as the ufo rays with different sizes to attack are satisfied. These checkpoints are placed into the canva to display the UFO attack to the pyramid.