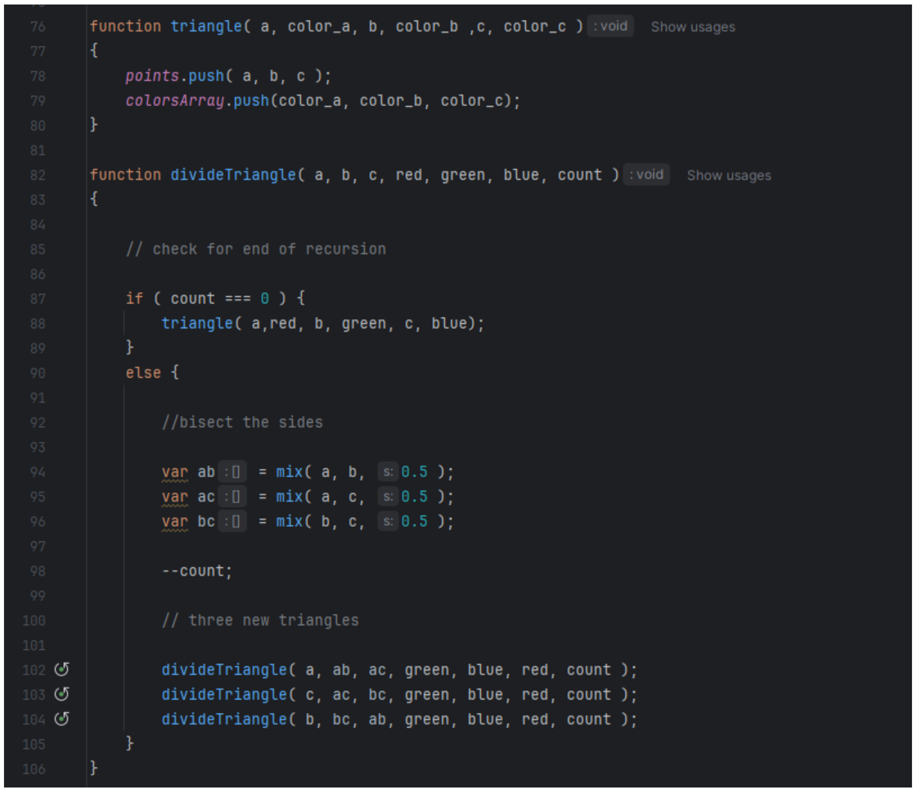
BBM414 Computer Graphics Lab. Programming Assignment #2 - WebGL2 Shape Drawing and Basic Shading

Muzaffer Berke Savaş
2220356044
Department of Computer Engineering
Hacettepe University
Ankara, Turkey
`b2220356044@cs.hacettepe.edu.tr`

Overview

The aim of this assignment is to dynamic color changes and center-based rotation of previous umbrella shape using rotation matrix and fragment shader features in WEBGL2.

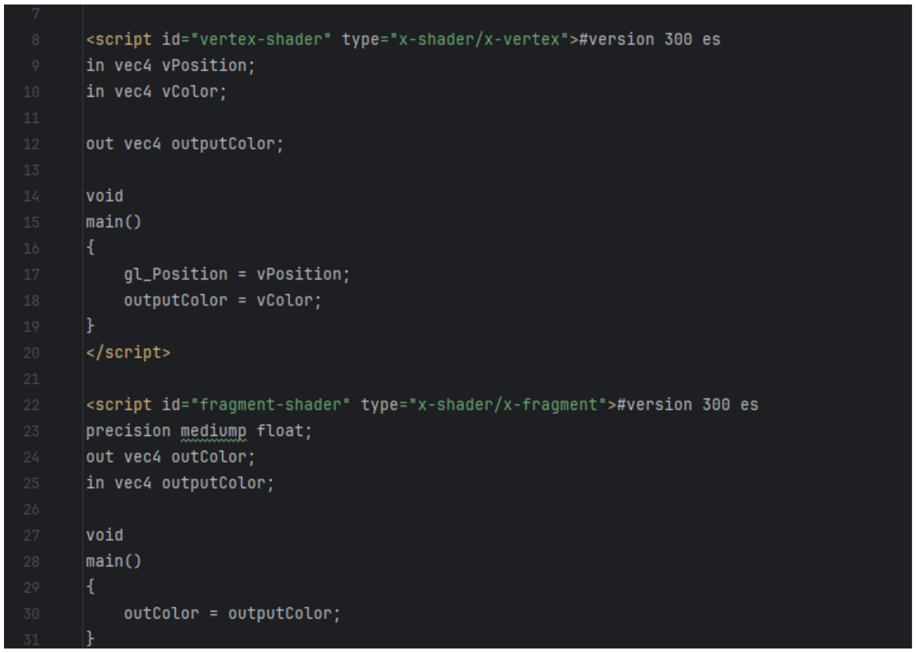
1 Part 1 - Modifying Color of the Triangles



```
function triangle( a, color_a, b, color_b ,c, color_c ) :void   Show usages
{
    points.push( a, b, c );
    colorsArray.push(color_a, color_b, color_c);
}

function divideTriangle( a, b, c, red, green, blue, count ) :void   Show usages
{

    // check for end of recursion

    if ( count === 0 ) {
        triangle( a,red, b, green, c, blue);
    }
    else {

        //bisect the sides

        var ab :[]  = mix( a, b,  s: 0.5 );
        var ac :[]  = mix( a, c,  s: 0.5 );
        var bc :[]  = mix( b, c,  s: 0.5 );

        --count;

        // three new triangles

        divideTriangle( a, ab, ac, green, blue, red, count );
        divideTriangle( c, ac, bc, green, blue, red, count );
        divideTriangle( b, bc, ab, green, blue, red, count );
    }
}
```

Figure 1

The triangle function adds given vertices and their colors to points and colorsArray, forming a colored triangle. The divideTriangle function subdivides a triangle recursively by calculating midpoints of its sides and creating smaller triangles until the recursion depth (count) reaches 0, at which point triangle() is called to render the final triangles in Figure 1.



```

<script id="vertex-shader" type="x-shader/x-vertex">#version 300 es
in vec4 vPosition;
in vec4 vColor;

out vec4 outputColor;

void
main()
{
    gl_Position = vPosition;
    outputColor = vColor;
}
</script>

<script id="fragment-shader" type="x-shader/x-fragment">#version 300 es
precision mediump float;
out vec4 outColor;
in vec4 outputColor;

void
main()
{
    outColor = outputColor;
}
```

Figure 2

The vertex shader takes `vPosition` (vertex position) and `vColor` (vertex color) as inputs and passes the position to `gl_Position` for rendering and the color to `outputColor` to be sent to the fragment shader. The fragment shader receives `outputColor` and sets it to `outColor`, which determines the final color output for each pixel in Figure 2.

2 Part 2 - Rotation of Umbrella and Changing Color of It Dynamically

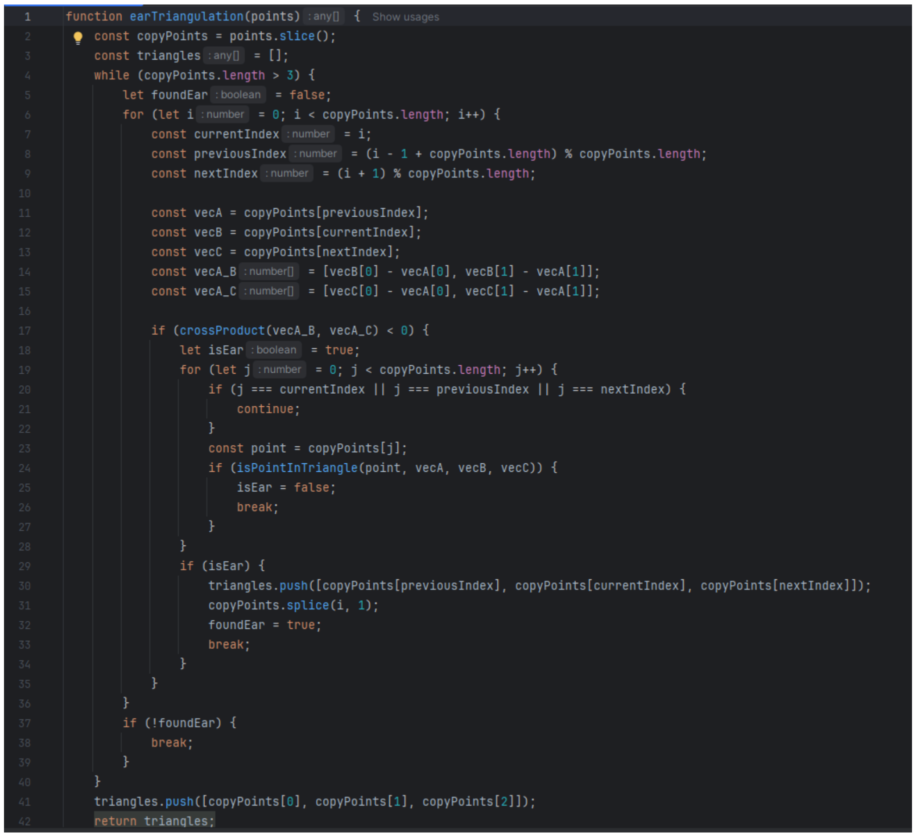
2.1 Revisiting Triangulation: A Shift to the Ear Clipping Method

Instead of using the previous triangulation method, we applied the **Ear Clipping Triangulation** algorithm, which divides a polygon into triangles by iteratively identifying and clipping "ears".

The function `earTriangulation()` works by:

1. Finding a convex triangle (ear) formed by three consecutive points, ensuring no other points lie inside it.
2. Clipping the ear and removing the middle vertex.
3. Repeating the process until only three points remain, forming the final triangle.

This method is efficient for polygon triangulation as it systematically reduces the polygon into smaller triangles by focusing on convex "ears".



```
function earTriangulation(points) {
    const copyPoints = points.slice();
    const triangles = [];
    while (copyPoints.length > 3) {
        let foundEar = false;
        for (let i = 0; i < copyPoints.length; i++) {
            const currentIndex = i;
            const previousIndex = (i - 1 + copyPoints.length) % copyPoints.length;
            const nextIndex = (i + 1) % copyPoints.length;

            const vecA = copyPoints[previousIndex];
            const vecB = copyPoints[currentIndex];
            const vecC = copyPoints[nextIndex];
            const vecA_B = [vecB[0] - vecA[0], vecB[1] - vecA[1]];
            const vecA_C = [vecC[0] - vecA[0], vecC[1] - vecA[1]];

            if (crossProduct(vecA_B, vecA_C) < 0) {
                let isEar = true;
                for (let j = 0; j < copyPoints.length; j++) {
                    if (j === currentIndex || j === previousIndex || j === nextIndex) {
                        continue;
                    }
                    const point = copyPoints[j];
                    if (isPointInTriangle(point, vecA, vecB, vecC)) {
                        isEar = false;
                        break;
                    }
                }
                if (isEar) {
                    triangles.push([copyPoints[previousIndex], copyPoints[currentIndex], copyPoints[nextIndex]]);
                    copyPoints.splice(i, 1);
                    foundEar = true;
                    break;
                }
            }
        }
        if (!foundEar) {
            break;
        }
    }
    triangles.push([copyPoints[0], copyPoints[1], copyPoints[2]]);
    return triangles;
```

Figure 3: Ear Clipping Algorithm

2.2 Center-Based Rotation of the Umbrella Model

```
const vertexShaderSource = ‘#version 300 es
    in vec2 aPosition;
    uniform mat3 uRotation;
    void main() {
        vec3 rotatedPosition = uRotation * vec3(aPosition, 1.0);
        gl_Position = vec4(rotatedPosition.xy, 0.0, 1.0);
    }‘;


let rotationEnabler = false;


canvas.addEventListener("mousemove", (event) => {
if (rotationEnabler) {
  const dx = event.clientX - centerX;
  const dy = event.clientY - centerY;
  const distance = Math.hypot(dx, dy);

  rotationSpeed = (distance / maxDistance) * 0.05 * (event.clientX > centerX ? 1 : -1);

}
});
```

```
const rMatrix = [
  Math.cos(angle) * scaleX , -Math.sin(angle) * scaleY, 0,
  Math.sin(angle) * scaleX , Math.cos(angle) * scaleY, 0,
  0,              0,                 1
];

gl.uniformMatrix3fv(rotationLocation, false, rMatrix);
```

For the above code fragment, the vertex shader takes each vertex position, applies the rotation matrix to it, and transforms the position accordingly. The rotation matrix is defined based on an angle that changes as the user moves the mouse. When the mouse moves over the canvas, the mousemove event calculates the distance between the current mouse position and the center of the umbrella, which then influences the rotation speed. The rotation speed is determined based on the distance from the center of the umbrella, with the direction of the rotation dependent on whether the mouse is to the left or right of the center. The uRotation uniform is updated with the new rotation matrix, and the vertices are transformed accordingly, causing the umbrella to rotate in response to user input.

2.3 Dynamic Fabric Color Adjustment of the Umbrella

```
const fragmentShaderSource = ‘#version 300 es
    precision mediump float;
    uniform vec4 uColor;
    out vec4 fragColor;
    void main() {
        fragColor = uColor;
    }‘;

let colorEnabler = false;
let colorInterval;
let currentColor = [1.0,0.0,0.0,1.0];

function start(){
    colorInterval = setInterval(() => {
    currentColor = [Math.random(), Math.random(), Math.random(), 1.0];
    }, 500);
}

function stop(){
  clearInterval(colorInterval);
}
```

The fragment shader is responsible for determining the color of each pixel rendered. It receives a uniform color variable uColor, which is updated in real-time.

The start function triggers a color change effect by using setInterval to randomly update the currentColor every 500 milliseconds. The color is randomly generated for the red, green, and blue components and passed to the fragment shader. This results in a continuous color shift in the umbrella's fabric. The stop function halts the color updates by clearing the interval, stopping the random color changes. The colorEnabler flag can be used to control whether this effect is enabled or disabled.

2.4 Reset Functionality: Returning the Umbrella to its Initial State

```
  window.addEventListener("keydown", (event) => {

  if (event.key === 'r'){
    stop();
    angle = 0;
    rotationSpeed = 0;
    rotationEnabler = false;
    currentColor = [1.0,0.0,0.0,1.0];
  }else if (event.key === 'm'){
      rotationEnabler = !rotationEnabler;
  } else if (event.key === 'c'){
    colorEnabler = !colorEnabler;
    if (colorEnabler) {
        start()
    }else {
      stop();
    }
  }
});
```

When the “r” key is pressed, the code resets the umbrella to its initial state. It performs the following actions:

- **Stops the color change**: The `stop()` function is called to halt any ongoing color transitions.
- **Resets rotation**: The rotation angle (`angle`) is set to 0, and the `rotationSpeed` is set to 0, stopping any rotation of the umbrella.
- **Disables rotation**: The `rotationEnabler` flag is set to `false`, ensuring that rotation is no longer active.
- **Resets color**: The umbrella’s fabric color is reset to the original red color, represented by the `currentColor` array $[1.0, 0.0, 0.0, 1.0]$.

In essence, pressing “r” returns the umbrella to its starting position and color, with rotation and color change effects paused.

References

[1] https://www.geeksforgeeks.org/how-to-use-shaders-to-apply-color-in-webgl/

[2] https://webgl2fundamentals.org/webgl/lessons/webgl-2d-rotation.html

[3] Two-Bit Coding. (2021). *Polygon Triangulation [1] - Overview of Ear Clipping*. YouTube. Published video.