EREN ÇAĞLAROĞLU CMPE360 HOMEWORK2 SEC03
11500168978

CMPE 360 HOMEWORK2

Introduction

A sample of JavaScript code intended for manipulating and combining 2D transformation matrices for objects in a 2D space is presented in this report. The two primary functions in the code are "GetTransform" and "ApplyTransform." A 3x3 transformation matrix is produced by the "GetTransform" function using user-specified scaling, rotation (in degrees), and translation parameters. Two transformation matrices can be composed in the desired order using the "ApplyTransform" function. These functions are essential for 2D graphics and animations in JavaScript applications since they may be used to do operations like scaling, rotating, translating, and chaining multiple transformations. The physics and real-world applications of the code are covered in greater detail in this report.

Functions

GetTransform Function

Position (X and Y), rotation (in degrees), and scale are the parameters that this method uses to produce a 3x3 transformation matrix. Scaling, rotation, and translation are the order in which the transformation is applied. Below is a summary of the functions of the code:

1.Create a 3x3 identity matrix by initializing it with ones on the major diagonal and zeros everywhere else.

2.Update the matrix's elements at indices 0 and 4 to the "scale" value to apply the scaling transformation. As a result, the item is scaled consistently in the X and Y directions.

3.Change the rotation angle's degree value to a radian. Next, determine the angle's sine and cosine.

4.Utilizing the cosine and sine values that were computed, apply the rotation transformation. To account for the rotation, the matrix entries at indices 0, 1, 3, and 4 are modified.

5.Update the elements at indices 6 and 7 with the "positionX" and "positionY" values, respectively, to apply the translation transformation.

6.Lastly, provide back an array representing the transformed transformation matrix.

```
function GetTransform(positionX, positionY, rotation, scale) {
    // Initialize the identity matrix
    const matrix : number[]  = [1, 0, 0, 0, 1, 0, 0, 0, 1];

    // Apply scale transformation
    matrix[0] = scale;
    matrix[4] = scale;

    // Convert rotation degrees to radians
    const angle : number  = (rotation * Math.PI) / 180;
    const cos : number  = Math.cos(angle);
    const sin : number  = Math.sin(angle);

    // Apply rotation transformation
    const a : number  = matrix[0];
    const b : number  = matrix[1];
    matrix[0] = a * cos - b * sin;
    matrix[1] = a * sin + b * cos;
    const c : number  = matrix[3];
    const d : number  = matrix[4];
    matrix[3] = c * cos - d * sin;
    matrix[4] = c * sin + d * cos;

    // Apply translation transformation
    matrix[6] = positionX;
    matrix[7] = positionY;

    // Return the resulting transformation matrix
    return matrix;
}
```

ApplyTransform Function

This function creates a new transformation matrix by combining the two input transformation matrices ("trans1" and "trans2"). This is what the code accomplishes:

1.As with GetTransform Function, initialize a result matrix as the identity matrix.

2.The matrices "trans1" and "trans2" should be multiplied in the proper sequence (trans1 * trans2). A nested loop structure is used to carry out the multiplication, and the elements of the output matrix are determined by adding the products of the appropriate components from "trans1" and "trans2."

3.The result matrix gets filled with the correct values as the loop iterates over each element of the 3x3 matrices.

4.Next, the transformation matrix that was produced is given back.

These two functions can be used to combine several transformations in the desired sequence and apply different transformations, including translation, rotation, and scaling, to objects in a 2D space.

```
// TO DO 2:Provides a 3x3 transformation matrix represented as an array containing 9 values arranged in column-major sequence.
// The inputs consist of transformation matrices following the identical format.
// The resulting transformation initially employs trans1 and subsequently applies trans2.
1 usage
function ApplyTransform(trans1, trans2) {
    // Initialize the result matrix as the identity matrix
    const result : number[]  = [1, 0, 0, 0, 1, 0, 0, 0, 1];

    // Multiply the matrices in the correct order (trans1 * trans2)
    for (let i : number  = 0; i < 9; i++) {
        const row : number  = Math.floor( x: i / 3);
        const col : number  = i % 3;
        let sum : number  = 0;
        for (let j : number  = 0; j < 3; j++) {
            sum += trans1[row * 3 + j] * trans2[j * 3 + col];
        }
        result[i] = sum;
    }

    // Return the resulting transformation matrix
    return result;
}
```

Conclusion

I have learned how to work with and apply 2D transformation matrices in JavaScript. The "GetTransform" function helped me understand how to scale, rotate, and translate objects. I also learned how to use the "ApplyTransform" function to combine these transformations into more intricate processes.With uses in web development, game design, and interactive simulations, this experience offers a basic tool for projects involving 2D graphics and animations. It emphasizes how crucial it is to use mathematical concepts when programming and how structured code is essential to accomplishing intricate visual effects.Along with providing me with useful solutions, this educational experience has opened my eyes to new possibilities for 2D animation and graphics research and development.