MANİSA
CELAL BAYAR
UNIVERSITY

CSE 3114 / CSE 3219

COMPUTER GRAPHICS

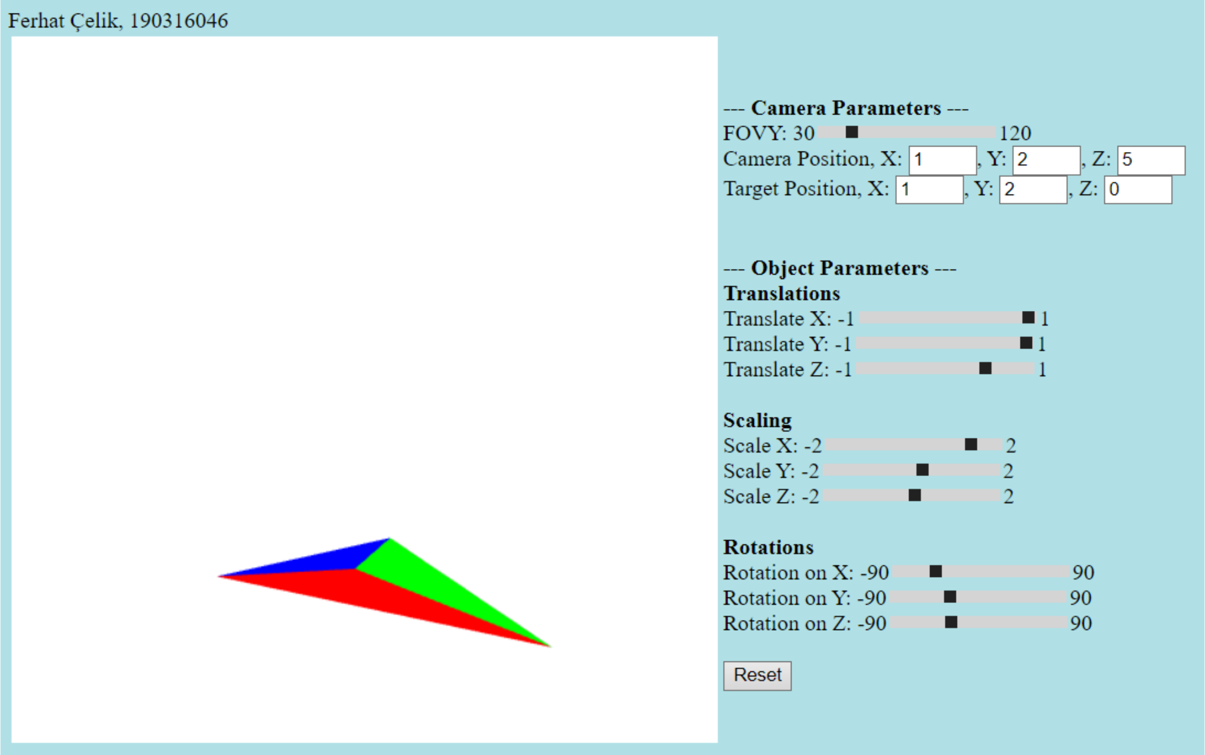
SPRING 2023

Homework #3 Report

Ferhat ÇELİK - 190316046

Submission Date: *21 May 2023*

Program Output



Ferhat Çelik, 190316046

--- **Camera Parameters** ---

FOVY: 30 120

Camera Position, X: 1 , Y: 2 , Z: 5

Target Position, X: 1 , Y: 2 , Z: 0

--- **Object Parameters** ---

Translations

Translate X: -1 1

Translate Y: -1 1

Translate Z: -1 1

Scaling

Scale X: -2 2

Scale Y: -2 2

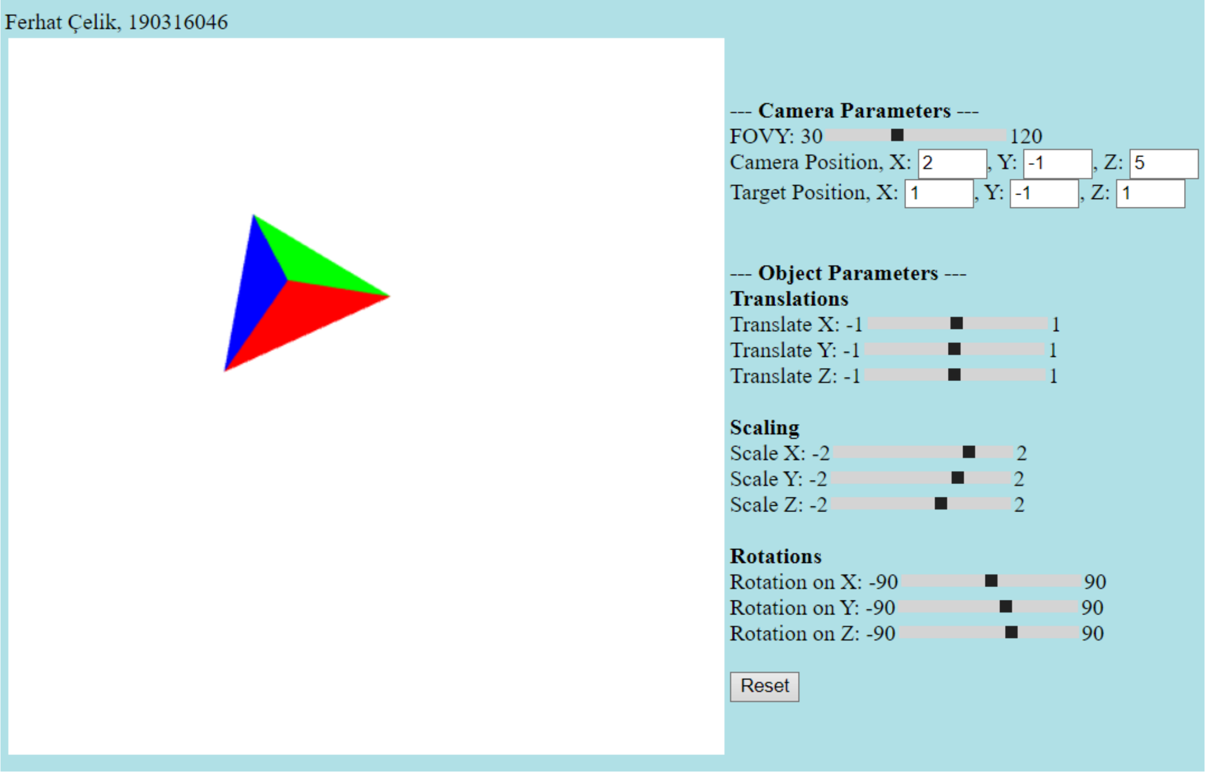
Scale Z: -2 2

Rotations

Rotation on X: -90 90

Rotation on Y: -90 90

Rotation on Z: -90 90



Ferhat Çelik, 190316046

--- **Camera Parameters** ---

FOVY: 30 120

Camera Position, X: 2 , Y: -1 , Z: 5

Target Position, X: 1 , Y: -1 , Z: 1

--- **Object Parameters** ---

Translations

Translate X: -1 1

Translate Y: -1 1

Translate Z: -1 1

Scaling

Scale X: -2 2

Scale Y: -2 2

Scale Z: -2 2

Rotations

Rotation on X: -90 90

Rotation on Y: -90 90

Rotation on Z: -90 90

Reflections

First, the codes provided to us showed the opposite face of tetrahedron, and it was bigger than I needed, so I multiplied scale x and scale y with 0.5, and scale z with -0.5 in render function. Which resulted in a smaller object and front face of tetrahedron. After some time, I used lookAt function and realized that I never had to do multiplications that I have done. So, I removed them.

Source Code

```
/** @type {WebGLRenderingContext} */
var canvas;
var gl;

//initial object transformations
var rotX = rotY = rotZ = 0;
var posX = posY = posZ = 0;
var scaleX = scaleY = scaleZ = 1;

//4 vertices to define tetrahedron corners
var vertices = [
    vec3(  0.0000,  0.0000, 1.0000 ),
    vec3(  0.0000,  0.9428, -0.3333 ),
    vec3( -0.8165, -0.4714, -0.3333 ),
    vec3(  0.8165, -0.4714, -0.3333 )
];

//colors of each tetrahedron corner
var vertexColors = [
    vec4( 0.0, 0.0, 1.0, 1.0 ),  // blue
    vec4( 1.0, 0.0, 0.0, 1.0 ),  // red
    vec4( 1.0, 1.0, 0.0, 1.0 ),  // yellow
    vec4( 0.0, 1.0, 0.0, 1.0 ),  // green
];

//initial camera and view parameters
var near = 0.3; //near clipping plane
var far = 11.0; //far clipping plane
var eyeX = 0; //camera position x
var eyeY = 0; //camera position y
var eyeZ = 5; //camera position z
var tarX = tarY = tarZ = 0; //camera target (at) position x, y, z
var fovy = 45.0;  // Field-of-view in Y direction angle (in degrees)
var aspect = 1.0; // Viewport aspect ratio
const up = vec3(0.0, 1.0, 0.0); //camera up vector

var modelViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;
var points = [];
var colors = [];

//function that generates tetrahedron geometry
function tetrahedron()
{
    points.push(vertices[0]);
    colors.push(vertexColors[0]);
    points.push(vertices[1]);
    colors.push(vertexColors[0]);
    points.push(vertices[2]);
    colors.push(vertexColors[0]);

    points.push(vertices[3]);
    colors.push(vertexColors[1]);
    points.push(vertices[0]);
    colors.push(vertexColors[1]);
    points.push(vertices[2]);
    colors.push(vertexColors[1]);

    points.push(vertices[1]);
    colors.push(vertexColors[2]);
    points.push(vertices[2]);
    colors.push(vertexColors[2]);
    points.push(vertices[3]);
    colors.push(vertexColors[2]);

    points.push(vertices[3]);
    colors.push(vertexColors[3]);
    points.push(vertices[1]);
    colors.push(vertexColors[3]);
    points.push(vertices[0]);
    colors.push(vertexColors[3]);
}

window.onload = function init() {
    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    gl.viewport( 0, 0, canvas.width, canvas.height );

    aspect =  canvas.width/canvas.height;

    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
    gl.enable(gl.DEPTH_TEST); //enable depth test for occlusion handling

    tetrahedron();//compute geometry

    //  Load shaders
    var program = initShaders( gl, "vertex-shader", "fragment-shader" );
    gl.useProgram( program );

    //initialize attribute buffers
    var vBuffer = gl.createBuffer();
    gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer);
    gl.bufferData( gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW );

    var vPosition = gl.getAttribLocation( program, "vPosition" );
    gl.vertexAttribPointer( vPosition, 3, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vPosition );

    var cBuffer = gl.createBuffer();
    gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer);
    gl.bufferData( gl.ARRAY_BUFFER, flatten(colors), gl.STATIC_DRAW );

    var vColor = gl.getAttribLocation( program, "vColor" );
    gl.vertexAttribPointer( vColor, 4, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vColor);

    // get uniform matrix locations
    modelViewMatrixLoc = gl.getUniformLocation( program, "modelViewMatrix" );
    projectionMatrixLoc = gl.getUniformLocation( program, "projectionMatrix"
);

    //____ Callback functions ____

    // sliders for viewing parameters
    document.getElementById("fovy").oninput = function(event) {
        fovy = parseFloat(event.target.value);
        projectionMatrix = perspective(fovy, aspect, near, far);
    };
    document.getElementById("tarX").onchange = function(event) {
        tarX = parseFloat(event.target.value);
    };
    document.getElementById("tarY").onchange = function(event) {
        tarY = parseFloat(event.target.value);
    };
    document.getElementById("tarZ").onchange = function(event) {
        tarZ = parseFloat(event.target.value);
    };
    document.getElementById("camX").onchange = function(event) {
        eyeX = parseFloat(event.target.value);
    };
    document.getElementById("camY").onchange = function(event) {
        eyeY = parseFloat(event.target.value);
    };
    document.getElementById("camZ").onchange = function(event) {
        eyeZ = parseFloat(event.target.value);
    };

    // sliders for object parameters
    document.getElementById("rotX").oninput = function(event) {
        rotX = parseFloat(event.target.value);
    };
    document.getElementById("rotY").oninput = function(event) {
        rotY = parseFloat(event.target.value);
    };
    document.getElementById("rotZ").oninput = function(event) {
        rotZ = parseFloat(event.target.value);
    };
    document.getElementById("posX").oninput = function(event) {
        posX = parseFloat(event.target.value);
    };
    document.getElementById("posY").oninput = function(event) {
        posY = parseFloat(event.target.value);
    };
    document.getElementById("posZ").oninput = function(event) {
        posZ = parseFloat(event.target.value);
    };
    document.getElementById("scaleX").oninput = function(event) {
        scaleX = parseFloat(event.target.value);
    };
    document.getElementById("scaleY").oninput = function(event) {
        scaleY = parseFloat(event.target.value);
    };
    document.getElementById("scaleZ").oninput = function(event) {
        scaleZ = parseFloat(event.target.value);
    };

    //reset button callback
    document.getElementById("ResetButton").addEventListener("click",
function(){
        rotX = rotY = rotZ = 0;
        posX = posY = posZ = 0;
        scaleX = scaleY = scaleZ = 1;
        fovy = 45.0;
        tarX = tarY = tarZ = 0;
        eyeX = eyeY = 0;
        eyeZ = 5;
        projectionMatrix = perspective(fovy, aspect, near, far);
        document.getElementById("fovy").value = fovy;
        document.getElementById("tarX").value = tarX;
        document.getElementById("tarY").value = tarY;
        document.getElementById("tarZ").value = tarZ;
        document.getElementById("camX").value = eyeX;
        document.getElementById("camY").value = eyeY;
        document.getElementById("camZ").value = eyeZ;
        document.getElementById("rotX").value = rotX;
        document.getElementById("rotY").value = rotY;
        document.getElementById("rotZ").value = rotZ;
        document.getElementById("posX").value = posX;
        document.getElementById("posY").value = posY;
        document.getElementById("posZ").value = posZ;
        document.getElementById("scaleX").value = scaleX;
        document.getElementById("scaleY").value = scaleY;
        document.getElementById("scaleZ").value = scaleZ;
    });

    render();
}

var render = function(){
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    var eye = vec3(eyeX, eyeY, eyeZ);
    var at = vec3(tarX, tarY, tarZ);
    var up = vec3(0.0, 1.0, 0.0);
    modelViewMatrix = lookAt(eye, at , up);
    modelViewMatrix = mult(modelViewMatrix, translate(posX, posY, posZ));
    modelViewMatrix = mult(modelViewMatrix, rotate(rotX, [1, 0, 0]));
    modelViewMatrix = mult(modelViewMatrix, rotate(rotY, [0, 1, 0]));
    modelViewMatrix = mult(modelViewMatrix, rotate(rotZ, [0, 0, 1]));
    modelViewMatrix = mult(modelViewMatrix, scalem(scaleX, scaleY, scaleZ));

    projectionMatrix = perspective(fovy, aspect, near, far);
    gl.uniformMatrix4fv( modelViewMatrixLoc, false, flatten(modelViewMatrix)
);
    gl.uniformMatrix4fv( projectionMatrixLoc, false, flatten(projectionMatrix)
);

    //draw the geometry
    gl.drawArrays( gl.TRIANGLES, 0, points.length );

    requestAnimFrame(render);
}
```