CG Assignment2

Seo, Myunggun
ms9144

1 Summary

Score: 20/20

2 Draw cube

Very well written.

Cube.html

```
<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
uniform float x_theta;
uniform float y_theta;
uniform float z_theta;
attribute vec4 vColor;
varying vec4 fColor;

void main(){
        float x = vPosition.x;
        float y = vPosition.y;
        float z = vPosition.z;

        float x1 = x*cos(z_theta) - y*sin(z_theta);
        float y1 = x*sin(z_theta) + y*cos(z_theta);
        float z1 = z;

        float x2 = x1;
        float y2 = y1*cos(x_theta) - z1*sin(x_theta);
        float z2 = y1*sin(x_theta) + z1*cos(x_theta);

        float z3 = z2*cos(y_theta) - x2*sin(y_theta);
        float x3 = z2*sin(y_theta) + x2*cos(y_theta);
        float y3 = y2;
        gl_Position = vec4(x3, y3, z3, 1);
        fColor = vColor;
}
</script>
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;
varying vec4 fColor;

void main(){
        gl_FragColor = fColor;
```

```
}
</script>
```

Cube.js

```
function(global) {
        "use strict";
        var gl; // global variable
        var num_vertices;
        var vertices;
        var ux, uy, uz; //location of angle of rotation around x, y, z axis

        window.onload = function init() {
                // Set up WebGL
                var canvas = document.getElementById("gl-canvas");
                gl = WebGLUtils.setupWebGL( canvas );
                if(!gl){alert("WebGL setup failed!");}

                // Clear canvas
                gl.clearColor(0.0, 0.0, 0.0, 1.0);
                gl.clear(gl.COLOR_BUFFER_BIT);

                // enable depth test
                gl.enable(gl.DEPTH_TEST);
                gl.depthFunc(gl.LEQUAL);
                gl.clearDepth(1.0);

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

                // Define length, set position of vertices
                var s = 0.5;
                var a = vec4(s, s, s);
                var b = vec4(-s, s, s);
                var c = vec4(-s, -s, s);
                var d = vec4(s, -s, s);
                var e = vec4(s, s, -s);
                var f = vec4(-s, s, -s);
                var g = vec4(-s, -s, -s);
                var h = vec4(s, -s, -s);
                vertices = dedim([quad(a,b,c,d), quad(f,e,h,g), quad(b,a,e,f),
                   quad(c,b,f,g), quad(d,c,g,h), quad(a,d,h,e)]);

                var vBuffer = gl.createBuffer();
                gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
                gl.bufferData(gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW);

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

                //Angle of rotation around each axis
```

```
            ux = gl.getUniformLocation(program, "x_theta");
            uy = gl.getUniformLocation(program, "y_theta");
            uz = gl.getUniformLocation(program, "z_theta");

            // Define colors, set color of vertices
            var R = vec3(1.0, 0.0, 0.0);
            var G = vec3(0.0, 1.0, 0.0);
            var B = vec3(0.0, 0.0, 1.0);
            var C = vec3(0.0, 1.0, 1.0);
            var M = vec3(1.0, 0.0, 1.0);
            var Y = vec3(1.0, 1.0, 0.0);
            var colors = dedim([quad(R), quad(G), quad(B), quad(C), quad(M),
               quad(Y)]);

            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, 3, gl.FLOAT, false, 0, 0);
            gl.enableVertexAttribArray(vColor);


            //Draw
            num_vertices = vertices.length/4;
            requestAnimationFrame(render);

    };

    function render(now){
            requestAnimationFrame(render);

            //at every frame, update the rotation value according to th evalue
                on the slider
            gl.uniform1f(ux, radians(document.getElementById('x-slider').value
               ));
            gl.uniform1f(uy, radians(document.getElementById('y-slider').value
               ));
            gl.uniform1f(uz, radians(document.getElementById('z-slider').value
               ));

            //
            gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
            gl.drawArrays(gl.TRIANGLES, 0, num_vertices);
    }
})(this);
```

3 L-System

No Comments. Well written code

L-Systems.js

```
(function(global) {
```

```
"use strict";
var gl; // global variable
var cos = Math.cos, sin = Math.sin, PI = Math.PI;

window.onload = function init() {
        // Set up WebGL
        var canvas = document.getElementById("gl-canvas");
        gl = WebGLUtils.setupWebGL( canvas );
        if(!gl){alert("WebGL setup failed!");}

        // Clear canvas
        gl.clearColor(0.0, 0.0, 0.0, 1.0);
        gl.clear(gl.COLOR_BUFFER_BIT);

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

        // Settings for Turtle
        var initial_config = {
                x: 0,
                y:0,
                theta: 0
        };
        var alpha = PI/8;
        var axiom = "F";
        var production_rules = {
                // F: "+X-Y-X+", X: "f--F--f", Y: "F"
                // F: "F-F++F-F"                //axiom:F, alpha = pi/3
                // F: "F[+F]F[-F]F"             //axiom:F, alpha = pi/8
                // F: "FF+F+F+FF+F+F-F"         //axiom:F, alpha = pi/2
                // F: "F+F-F-FF+F+F-F"          //axiom:F, alpha = pi/2
                F: "FF+[+F-F-F]-[-F+F+F]" //axiom:F, alpha = pi/8
                // F: "FF", X:"F[+X]F[-X]+X" //axiom:X, alpha = pi/9
        };
        var num_productions = 4;

        // Create Turtle
        var v = turtle(initial_config, alpha, axiom, production_rules,
           num_productions);

        // Load data into a buffer
        var vBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(v), gl.STATIC_DRAW
           );

        // Do shader plumbing
        var vPosition = gl.getAttribLocation(program, "vPosition");
        gl.vertexAttribPointer(vPosition, 2, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(vPosition);
```

```
        //Draw
        var num_vertices = v.length/2;
        gl.drawArrays(gl.LINES, 0, num_vertices);
};

//return array of (x,y) vertices
function turtle(initial_config, alpha, axiom, production_rules,
   num_productions){
        /**
         * STEP 1. Replace axiom with production rules
         */
        axiom = axiom.split('');
        var axiom_len, new_axiom;
        for (var production = 0; production < num_productions; production
           ++) {
                axiom_len = axiom.length;
                new_axiom = [];
                for (var i = 0, char; i < axiom_len; i++) {
                        char = axiom[i];
                        if (production_rules.hasOwnProperty(char)) {
                                new_axiom.push(production_rules[char].
                                   split(''));
                                new_axiom = dedim(new_axiom);
                        }
                        else {
                                new_axiom.push(char);
                        }
                }
                axiom = new_axiom;
        }
        global.axiom = axiom.join('');

        /**
         * STEP 2. Read through axiom, insert vertices
         */
        var config = clone(initial_config), config_save = [], result = [];
        for (var rule of axiom) {
                switch(rule) {
                        case 'f':
                                config.x += cos(config.theta);
                                config.y += sin(config.theta);
                                break;
                        case 'F':
                                result.push(config.x, config.y);
                                config.x += cos(config.theta);
                                config.y += sin(config.theta);
                                result.push(config.x, config.y);
                                break;
                        case '+':
                                config.theta += alpha;
                                break;
                        case '-':
                                config.theta -= alpha;
```

```
                        break;
                    case '[':
                        config_save.push(clone(config));
                        break;
                    case ']':
                        config = config_save.pop();
                        break;
                    default: break;
                }
            }
            normalize(result);
            return result;
        }
```