UNIVERSITI SAINS MALAYSIA

PUSAT PENGAJIAN SAINS KOMPUTER
(School of Computer Sciences)

SEMESTER I 2021/22

HELAIAN KULIT TUGASAN
(ASSIGNMENT COVER SHEET)

Nama *(Name)* Siti Zainab Binti Sahardin		Nombor Matrik *(Matric Number)* 147672
Masa/Kumpulan Tutorial (Jika Berkenaan) *(Tutorial Time/Group (If Applicable))* -	Tutor (Jika Berkenaan) *(Tutor (If Applicaple))* -	Tarikh Akhir Penyerahan Tugasan *(Date Assignment is Due)* 22 Dec 21 (online, before 11.59 pm)

Kod Kursus *(Course Code)* CPC354	Tajuk Kursus *(Course Title)* *COMPUTER GRAPHICS & VISUALISATION*	No./Tajuk Tugasan *(Assignment No./Title)* TUGASAN I – PENGENALAN KEPADA PENGATURCARAAN GRAFIK & WebGL *(ASSIGNMENT I –INTRODUCTION TO GRAPHICS PROGRAMMING & WebGL)*
Diterima Oleh: Tandatangan Staf *(Received By: Staff Signature)*		Tarikh/Masa Diterima *(Date/Time Lodged)*

Pengisytiharan

Saya/Kami* mengesahkan bahawa tugasan ini adalah kerja saya/kami* seluruhnya kecuali apabila saya/kami* memberikan rujukan lengkap kepada kerja yang dilakukan oleh orang lain, dan bahan di dalam tugasan ini tidak pernah diserahkan untuk penilaian dalam mana-mana kursus pengajian formal yang lain.

Declaration

I/We certify that this assignment is entirely my/our* own work, except where I/we* have given fully documented references to the work of others, and that the material contained in this assignment has not previously been submitted for assessment in any formal course of study.*

*Potong mana yang tidak berkenaan (bergantung pada sama ada tugasan individu atau kumpulan).
Delete where applicable (depends on whether individual or group assignment).

Tandatangan Pelajar :
(Signature of Student)

Gred *(Grade)*	Pemeriksa *(Marker)* Dr. Nur Intan Raihana Ruhaiyem
Ulasan Penanda *(Marker's Comments)*	

✂----------

Resit Pelajar
(Student Receipt)

Untuk diisi oleh pelajar
(To be completed by the student)

Nama Pelajar *(Student Name)* Siti Zainab Binti Sahardin		Nama Pensyarah *(Name of Lecturer)* Dr. Nur Intan Raihana Ruhaiyem
Kod Kursus *(Course Code)* CPC354	Tajuk Kursus *(Course Title)* *COMPUTER GRAPHICS & VISUALISATION*	No./Tajuk Tugasan *(Assignment No./Title)* TUGASAN I – PENGENALAN KEPADA PENGATURCARAAN GARFIK & WebGL *(ASSIGNMENT I –INTRODUCTION TO GRAPHICS PROGRAMMING & WebGL)*
Diterima Oleh: Tandatangan Staf *(Received By: Staff Signature)*		Tarikh/Masa Diterima *(Date/Time Lodged)*

Description of Program

Figure 1 shows the initial view of the program.

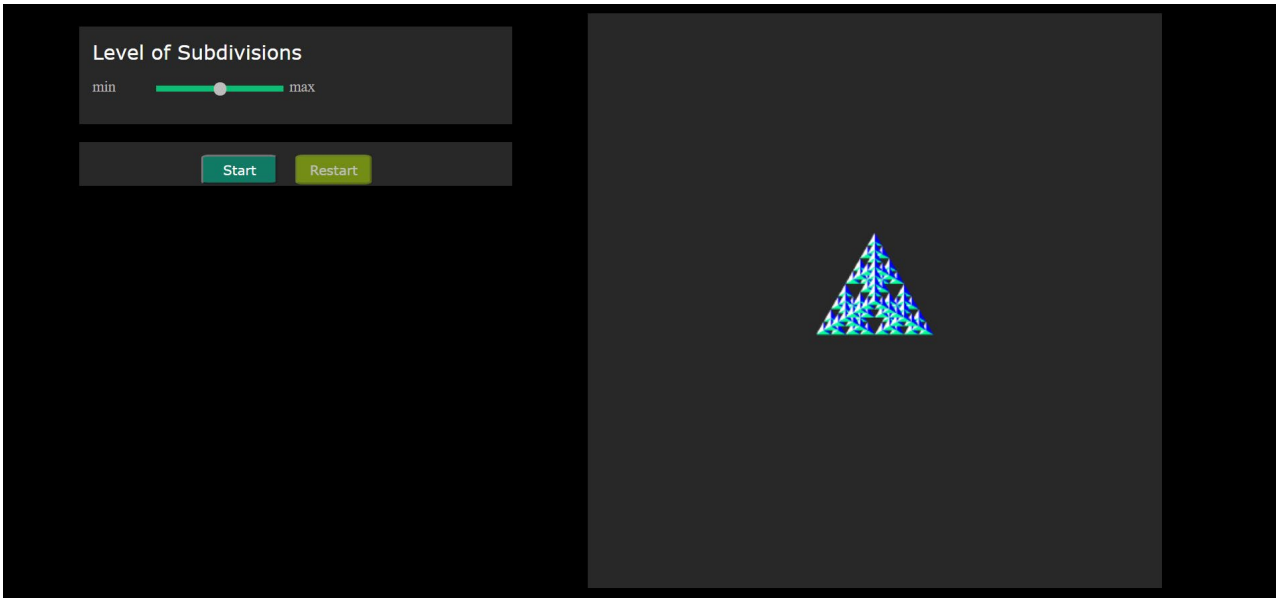

Figure 1

User can click the “Start” button to initialize the gasket movement. The Start button will change to “Stop” and the gasket will rotate at the centre of the canvas to the right 180 degrees and then back to its original orientation before rotating to the left 180 degrees and back to its original orientation. Next, it will enlarge to another size before returning back to the original size. After that, it will move around the canvas until the “Stop” button is pressed.

User also can change the number of times of gasket’s subdivision by sliding the slider “Level of Subdivisions”. Sliding to the right will increase the number of subdivision while decrease to another side.

TVIdent.html

```
<!DOCTYPE html>
<html lang="en-US">

<head>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
    <title>TV Ident {3D Sierpinski Gasket}</title>

    <script id="vertex-shader" type="x-shader/x-vertex">
    attribute vec4 vPosition;
    attribute vec4 vColor;
    varying vec4 fColor;
    uniform vec3 theta;
    uniform float scale;
    uniform vec2 trans;
```

```
    void main() {
        vec3 angles = radians(theta);
        vec3 c = cos(angles);
        vec3 s = sin(angles);

        mat4 rx = mat4(1.0, 0.0, 0.0, 0.0,
                       0.0, c.x, s.x, 0.0,
                       0.0, -s.x, c.x, 0.0,
                       0.0, 0.0, 0.0, 1.0);

        mat4 ry = mat4(c.y, 0.0, -s.y, 0.0,
                       0.0, 1.0, 0.0, 0.0,
                       s.y, 0.0, c.y, 0.0,
                       0.0, 0.0, 0.0, 1.0);

        mat4 rz = mat4(c.z, s.z, 0.0, 0.0,
                       -s.z, c.z, 0.0, 0.0,
                       0.0, 0.0, 1.0, 0.0,
                       0.0, 0.0, 0.0, 1.0);

        mat4 rotationMat = rz * ry * rx;

        mat4 scalingMat = mat4(scale, 0.0, 0.0, 0.0,
                               0.0, scale, 0.0, 0.0,
                               0.0, 0.0, scale, 0.0,
                               0.0, 0.0, 0.0, 1.0);

        mat4 transMat = mat4(1.0, 0.0, 0.0, 0.0,
                             0.0, 1.0, 0.0, 0.0,
                             0.0, 0.0, 1.0, 0.0,
                             trans[0], trans[1], 0.0, 1.0);

        gl_Position = transMat * scalingMat * rotationMat * vPosition;
        fColor = vColor;
    }
    </script>

    <script id="fragment-shader" type="x-shader/x-fragment">
    precision mediump float;
    varying vec4 fColor;

    void main() {
        gl_FragColor = fColor;
    }
    </script>
    <script type="text/javascript" src="Common/webgl-utils.js"></script>
    <script type="text/javascript" src="Common/initShaders.js"></script>
    <script type="text/javascript" src="Common/MV.js"></script>
    <script type="text/javascript" src="TVIdent.js"></script>
    <link rel="stylesheet" href="pageStyle.css">

</head>

<body>
```

```
    <div>
        <div id="menu">
            <div id="propertybox">
                <label for="division">Level of Subdivisions </label>
                <p>min<input type="range" id="division" class="slider settings"
name="division" min="1" max="5" step="1"
                        value="3" />  max</p>
            </div>

            <div id="propertybox">
                <input type="button" id="start-button" value="Start">
                <input type="button" id="restart-button" value="Restart"><br>
            </div>
            <br>
        </div>

        <canvas id="gl-canvas" width="512" height="512">
            Oops ... your browser doesn't support the HTML5 canvas element
        </canvas>
    </div>
</body>

</html>
```

TVIdent.js

```
var canvas;
var gl;
var points = [];
var colors = [];

var baseColors = [
  vec4(1.0, 1.0, 1.0, 1.0), //white
  vec4(0.0, 1.0, 0.6, 1.0), //mint green
  vec4(0.0, 0.0, 1.0, 1.0), //blue
  vec4(0.0, 1.0, 0.0, 1.0), //green
];

window.onload = function init() {
    // WebGL functions
    canvas = document.getElementById("gl-canvas");
    gl = WebGLUtils.setupWebGL(canvas);

      if (!gl) {
        alert("WebGL isn't available");
    }
    gl.viewport(0, 0, canvas.width, canvas.height);
    gl.clearColor(0.157, 0.157, 0.157, 1.0);
    gl.enable(gl.DEPTH_TEST);
    const program = initShaders(gl, "vertex-shader", "fragment-shader");
    gl.useProgram(program);
```

```
    // shader controls
    const controls = {};
    controls.vColor = gl.getAttribLocation(program, "vColor");
    controls.vPosition = gl.getAttribLocation(program, "vPosition");
    controls.thetaLoc = gl.getUniformLocation(program, "theta");
    controls.scaleLoc = gl.getUniformLocation(program, "scale");
    controls.transLoc = gl.getUniformLocation(program, "trans");

    // 3D gasket properties
    const gasket = {
        vertices: [ vec3(0.0, 0.0, -0.25),
                    vec3(0.0, 0.2357, 0.0833),
                    vec3(-0.2041, -0.1179, 0.0833),
                    vec3(0.2041, -0.1179, 0.0833),],
        division: 3,
        speed: 200,
        theta: [0, 0, 0],
        degree: 180,
        rotateXYZ: [false, false, true],
        scale: 1,
        scaleFac: 3,
        trans: [0.0, 0.0],
        transMode: 0,
        pause: true,
    };

    // animation list for 3D gasket
    const animsRegistry = obj => [
    // rotation Z (default)
    rotation.bind(null, obj, -obj.degree, 2),
    rotation.bind(null, obj, obj.degree, 2),
    rotation.bind(null, obj, 0, 2),

    // enlarge and shrink
    scaling.bind(null, obj, obj.scaleFac),
    scaling.bind(null, obj, obj.scale),

    // random hit and bounce
    setDelta.bind(null, obj),
    translation.bind(null, obj),
    ];

    // settings for 3D gasket
    const settings = Array.from(document.querySelectorAll(".settings"));
    settings.forEach(setting => {
        setting.addEventListener("change", () => {
            gasket[setting.name] = Number(setting.value);
            let textbox =
document.querySelector(`[class="textbox"][name="${setting.name}"]`);

            if (textbox !== null) {
                textbox.value = setting.value;
            }
```

```
            renderObject(controls, gasket);
            gasket.anims = animsRegistry(gasket);
            gasket.currentAnim = gasket.anims.shift();
        });
    });

    const checkboxes =
Array.from(document.querySelectorAll('input[type="checkbox"]'));
    checkboxes.forEach((checkbox, i) => {
        checkbox.checked = false;
        checkbox.addEventListener("change", e => {
            gasket.rotateXYZ[i] = e.target.checked;
        });
    });

    const inputs = settings.concat(checkboxes);

    const startBtn = document.getElementById("start-button");
    startBtn.addEventListener("click", () => {
        if (!gasket.pause) {
            gasket.pause = true;
            startBtn.value = "Start";
            startBtn.style.background = "#117A65";
        } else {
            gasket.pause = false;
            animate(gasket, controls);
            inputs.forEach(i => {i.disabled = true;});
            startBtn.value = "Stop";
            startBtn.style.background = "#B03A2E";
        }
    });

    restartBtn = document.getElementById("restart-button"); // global var
    restartBtn.disabled = true;
    restartBtn.addEventListener("click", () => {
        gasket.pause = true;
        gasket.theta = [0, 0, 0];
        gasket.trans = [0.0, 0.0];
        renderObject(controls, gasket);
        gasket.anims = animsRegistry(gasket);
        gasket.currentAnim = gasket.anims.shift();
        inputs.forEach(i => {i.disabled = false;});
        restartBtn.disabled = true;
        startBtn.value = "Start";
        startBtn.style.background = "#117A65";
    });


    // initial display of static 3D gasket
    renderObject(controls, gasket);
    // obtain animation list and start 3D gasket animation
    gasket.anims = animsRegistry(gasket);
    gasket.currentAnim = gasket.anims.shift();
};
```

```
function animate(obj, controls) {
    if (obj.pause === true)
        return;

    // enable restart button if it is the last animation (translation)
    if (obj.anims.length === 1)
        restartBtn.disabled = false;

    // current animation completes, switch animation
    if (obj.currentAnim()) {
        obj.currentAnim = obj.anims.shift(); // get first animation from list
    } else {
        // current animation has not completed, proceeds with same animation
        gl.uniform3fv(controls.thetaLoc, flatten(obj.theta));
        gl.uniform1f(controls.scaleLoc, obj.scale);
        gl.uniform2fv(controls.transLoc, obj.trans);
        gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
        gl.drawArrays(gl.TRIANGLES, 0, points.length);
    }
    requestAnimationFrame(() => animate(obj, controls));
}

function rotation(obj, degree, axis) {
    // if rotationX/Y/Z is enabled
    if (obj.rotateXYZ[axis] === true) {
        let difference = degree - obj.theta[axis];
        if (Math.abs(difference) > obj.speed * 0.01) {
            // add/subtract based on sign
            obj.theta[axis] += Math.sign(difference) * obj.speed * 0.01;
            return false;
        } else {
        obj.theta[axis] = degree;
        return true;
        }
    } else
        return true;
}

function scaling(obj, scaleFac) {
    let difference = scaleFac - obj.scale;
    if (Math.abs(difference) > obj.speed * 0.0005) {
        // add/subtract based on sign
        obj.scale += Math.sign(difference) * obj.speed * 0.0005;
        return false;
    } else {
        obj.scale = scaleFac;
        return true;
    }
}

function translation(obj) {
    // rotating, rotate about z axis
    if (obj.transMode === 1) {
```

```
        obj.theta[2] -= obj.speed * 0.01;
    } // dancing, rotate about y axis
    else if (obj.transMode === 2) {
        obj.theta[1] += obj.speed * 0.01;
    } // flipping. rotate about x axis
    else if (obj.transMode === 3) {
        obj.theta[0] += obj.speed * 0.01;
    } // paralysing, rotate about all axes
    else if (obj.transMode === 4) {
        // alternate between 2 directions
        if (Math.random() > 0.5) {
            obj.theta[0] += obj.speed * 0.01;
            obj.theta[1] += obj.speed * 0.01;
            obj.theta[2] -= obj.speed * 0.01;
        } else {
            obj.theta[0] -= obj.speed * 0.01;
            obj.theta[1] -= obj.speed * 0.01;
            obj.theta[2] += obj.speed * 0.01;
        }
    }
    // reverse x when any vertex hits left/right
    if (obj.vertices.some(v => Math.abs(v[0] + obj.trans[0] / obj.scale) > 0.97 /
obj.scale)) {
        obj.deltaX = -obj.deltaX;
    }

    // reverse y when any vertex hits top/bottom
    if (obj.vertices.some(v => Math.abs(v[1] + obj.trans[1] / obj.scale) > 0.97 /
obj.scale)) {
        obj.deltaY = -obj.deltaY;
    }

    obj.trans[0] += obj.deltaX;
    obj.trans[1] += obj.deltaY;
    return false;
}

// adjust delta (displacement) based on object's speed
function setDelta(obj) {
    obj.deltaX = obj.speed * Math.cos(Math.PI / 3) * 0.00004;
    obj.deltaY = obj.speed * Math.sin(Math.PI / 3) * 0.00004;
    return true;
}

// 3D gasket generation functions
function renderObject(controls, obj) {
    points = [];
    colors = [];
    divideTetra( obj.vertices[0],
                 obj.vertices[1],
                 obj.vertices[2],
                 obj.vertices[3],
                 obj.division);
```

```
    let cBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, flatten(colors), gl.STATIC_DRAW);
    gl.vertexAttribPointer(controls.vColor, 4, gl.FLOAT, false, 0, 0);
    gl.enableVertexAttribArray(controls.vColor);

    let vBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW);
    gl.vertexAttribPointer(controls.vPosition, 3, gl.FLOAT, false, 0, 0);
    gl.enableVertexAttribArray(controls.vPosition);

    gl.uniform3fv(controls.thetaLoc, flatten(obj.theta));
    gl.uniform1f(controls.scaleLoc, obj.scale);
    gl.uniform2fv(controls.transLoc, obj.trans);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    gl.drawArrays(gl.TRIANGLES, 0, points.length);
}

function triangle(a, b, c, color) {
    colors.push(baseColors[color]);
    points.push(a);
    colors.push(baseColors[color]);
    points.push(b);
    colors.push(baseColors[color]);
    points.push(c);
}

function tetra(a, b, c, d) {
    triangle(a, c, b, 0);
    triangle(a, c, d, 1);
    triangle(a, b, d, 2);
    triangle(b, c, d, 3);
}

function divideTetra(a, b, c, d, count) {

    if (count === 0) {  // check for end of recursion
        tetra(a, b, c, d);
    } else {    //bisect the sides
        let ab = mix(a, b, 0.5);
        let ac = mix(a, c, 0.5);
        let ad = mix(a, d, 0.5);
        let bc = mix(b, c, 0.5);
        let bd = mix(b, d, 0.5);
        let cd = mix(c, d, 0.5);

        --count;
        // four new triangles
        divideTetra(a, ab, ac, ad, count);
        divideTetra(ab, b, bc, bd, count);
        divideTetra(ac, bc, c, cd, count);
        divideTetra(ad, bd, cd, d, count);
    }
```

```
}
```

pageStyle.css

```
body {
  margin: 0px;
  padding: 0px;
  overflow: hidden;
  background: rgb(0, 0, 0);
}

#menu {
  position: absolute;
  left: 3vw;
  width: 37vw;
  height: 100vh;
  display: block;
  box-sizing: border-box;
  background-color: black;
  text-align: left;
  padding-left: 3vw;
  padding-top: 2vw;
}

canvas {
  grid-column-start: 2;
  grid-column-end: 3;
}

#propertybox {
  width: 512;
  height: 200;
  padding: 3%;
  margin-bottom: 3vh;
  background: rgb(40, 40, 40);
}

label {
  color: #ffffff;
  font-family: verdana, sans-serif;
  font-size: 20px;
}

p {
  color: #b9b4b4;
  font-family: "Times New Roman", Times, serif;
  font-size: 15px;
}

.colorlabel {
  margin: 5%;
}
```

```css
.slider {
  appearance: none;
  width: 10vw;
  height: 1vh;
  background-color: rgb(0, 255, 149);
  outline: none;
  opacity: 0.7;
}

.slider:hover {
  opacity: 1;
}

.slider:disabled {
  background-color: grey;
}

.slider::-webkit-slider-thumb {
  -webkit-appearance: none;
  appearance: none;
  width: 1vw;
  height: 1vw;
  border-radius: 50%;
  background: white;
  cursor: pointer;
}

#division {
  margin-left: 10%;
}

#scale {
  margin-left: 30%;
  margin-top: 5%;
}

#speed {
  margin-left: 9%;
}

#scaleFac {
  margin-left: 26%;
  margin-top: 4%;
}

#degree {
  margin-left: 19.2%;
  margin-top: 4%;
}

#rotateX,
#rotateY {
  margin-top: 3%;
  margin-left: 14%;
```

```
}

#transMode {
  margin-top: 3%;
  margin-left: 22%;
  width: 10vw;
  font-family: Verdana, sans-serif;
  background-color: rgb(160, 65, 123);
  color: white;
}

#start-button {
  width: 6vw;
  height: 5vh;
  background: #117a65;
  color: white;
  border-radius: 10%;
  position: absolute;
  left: 34%;
  font-family: Verdana, sans-serif;
}

#restart-button {
  width: 6vw;
  height: 5vh;
  background: #94b90e;
  color: white;
  border-radius: 10%;
  position: absolute;
  left: 54%;
  font-family: Verdana, sans-serif;
}

#start-button:hover {
  cursor: pointer;
  color: black;
  background: #27ae60;
}

#restart-button:enabled:hover {
  cursor: pointer;
  color: black;
  background: #77f312;
}

#restart-button:disabled {
  opacity: 0.7;
}

canvas {
  position: absolute;
  left: 45vw;
  width: 100vh;
  height: 100vh;
```

```
  display: block;
  box-sizing: border-box;
  border: 2vh solid black;
  background-color: rgb(30, 30, 30);
}
```