King Fahd University of Petroleum & Minerals
College of Computing and Mathematics

Information and Computer Science Department

ICS 415: Computer Graphics (3-0-3)

First Semester 2023-2024 (Term 231)

Due date: October 27, 2023 **Homework #** 02

Problem # 01

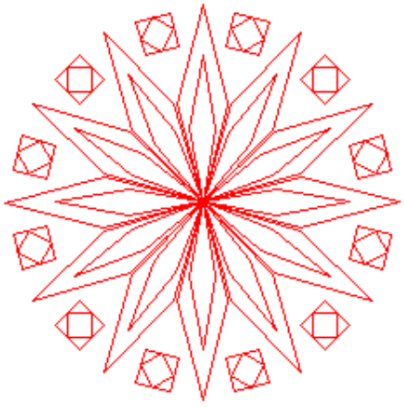
[50 POINTS]

In this problem, you will use WebGL and matrix functions provided in MV.js for implementing geometric transformations to construct a given design. Specifically, you will make use of the MV.js functions, translate(dx, dy, dz), rotate(angle, vx, vy, vz), and scalem(sx, sy, sz) to create the pattern shown below.

Note: Do not use the shear transformation. This makes the task much harder!

Recall that the translate(dx, dy, dz) function creates a translation matrix that, when multiplied with the model matrix, will translate all vertices in a direction and distance given by the vector (dx, dy, dz). rotate(angle, vx, vy, vz) creates a matrix that rotates the vertices by the specified angle (in degrees!) about an axis of rotation specified by the vector components vx, vy and vz. Finally, scalem(x, y, z) generates a matrix that scales the vertices by the factors sx, sy and sz (in the x, y and z directions, respectively).

Your task is to apply these transformations a square to create (approximately) the following pattern:



Choose the dimensions of each petal of the flower so that it looks as similar as possible to the flower shown above. There are two strict requirements:

- First, the center vertex of each petal must be positioned at the point (0, 0) in the clipping rectangle.
- Second, the diamonds that are inside the squares located around the outside must fit exactly in the square. In other words, you must calculate the side length of the diamond relative to the side length of the square so that each vertex of the diamond falls on the border of the square at the center of the square side.

Getting Started

- You must begin with the following matrix definitions:

  For the canvas size, use: <canvas id="gl-canvas" width="500" height="500">

- The viewport should also be 500 x 500 pixels.

- You must define only four vertices in your code, given as follows:

```
var vertices = [ ];     //This should be a global variable
...

var x=-0.1;    //Lower left corner
var y=-0.1;
var side = 0.2;

vertices.push(vec2(x, y));
vertices.push(vec2(x+side, y));
vertices.push(vec2(x+side, y+side));
vertices.push(vec2(x, y+side));
```

  These vertices produce a version of the following square:

- Without redefining any of these vertices or adding any more vertices, use the transformation matrices (translation, rotation and scale, but not shear) to transform the square and create the pattern. To get started, try to figure out how to create each of the petals from transforming the square. Decide what transformations you would use to create the following:

- Once you have the individual pieces, you need to transform and draw them in the proper orientations and positions. Try to make your code as compact as possible -- in particular, use loops to create parts of the design at different orientations.

What to submit for Problem # 01?

1. flower.html
2. flower.js
3. flower.png (image screenshot of your flower shape)

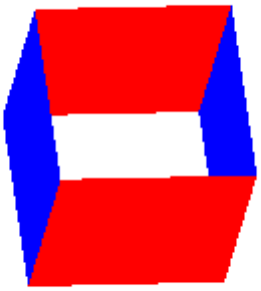
Problem # 02

[50 POINTS]

In this problem, you will use WebGL to create a 3-Dimensional object that rotates about different axes depending on a key-press. You will first create a simple 3D box, open at the top and bottom, then you will add the code that allows it to rotate. Finally, you will write a program that recursively divides each of the quadrilateral faces into two new quads, generating the approximation of a cylinder (that is open at the top and bottom).

Part a. A spinning box.

Write a program similar to the example program for a spinning 3D color cube from chapter 4 of the text (cube.html and cube.js). This program should create a box, with 4 quadrilateral faces that is open at the top and bottom (so you need to remove the top and bottom faces from the cube). Make your box centered on the origin. Make sure the faces of the box are created so that the "outward" face is toward the outside of the object. (Use the right-hand rule to make sure you list the vertices in the correct order to accomplish this). Alternate the colors of the sides between red and blue so that you can see the three-dimensionality of the box. You will need to change the VertexColors array to accomplish this. You should be able to use the buttons (that were already there from cube.html) to see the rotation of the box around different axes. Save this file as box.js and box.html. The following image shows the box viewed at an angle from the top:



Part b. Add a keypress option to the user interface

Save the program you wrote in part a into new files called, spin.js and spin.html. Modify this program to include a user-interface that detects a keypress:

- A press of the 'f' key should cause the box to spin about the x axis.
- The 'j' key should cause spin about the y axis.
- The 'k' key should cause spin about the z axis.

To respond to a keypress, use the function: window.onkeydown = keyResponse, where keyResponse() is the event listener for a keypress. An example of responding to a keypress is provided in the rotatingSquare program (available on the Blackboard). You can get the key value by using the following command:

```
var key = String.fromCharCode(event.keyCode);
```

Be aware that the string that is returned is a capitalized letter even if the user does not hold down the shift key.

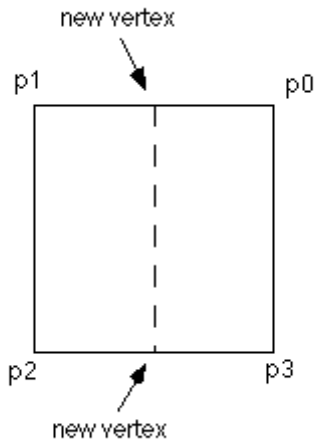
To make it so that the box only spins when a key is being held down, you should remove the angle increment (theta[axis] += 1.0;) from the render( ) function, and place it in each case of the switch( ) statement in the keyResponse( ) function. That is, for each key, you should change the axis of rotation and increment theta for that axis. If the user holds the key down, the keyResponse( ) function will be called repeatedly to show the spinning box. Save the code for this part in files called spin.html and spin.js.

Part c. Approximating a cylinder

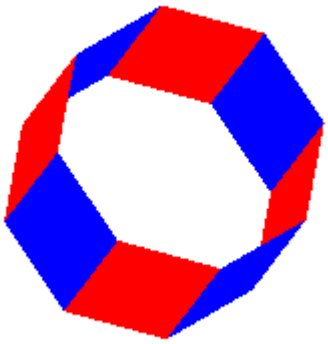
One way to generate a sphere in openGL, is by successively dividing triangular faces of a polyhedron into more triangles, and normalizing all the vertices so that they lie on the surface of a sphere. In this part of the problem, you will use a recursive strategy by using the divide_triangle function to recursively divide the object into triangular faces, you will write a recursive divide_quad() function that will recursively divide the quadrilateral faces of your box into 2 new quadrilateral faces.

Save the code from part b into two new files, called cylinder.js and cylinder.html. First, you will need to normalize the vertices of your box so that they all lie an equal distance from the y axis. If a vertex is at position (x, y, z), and you want it to be a distance, d, from the y axis, you can normalize by multiplying the x and z components by d/(sqrt(x*x + z*z)). Make sure none of your vertices is at the point (0, 0, 0) before doing this! Second, create a recursive function, divide_quad(), that takes four vertices and an integer counter as parameters. divide_quad() should calculate two new vertices

from the four given as parameters, one at the center of the top edge and one at the center of the bottom edge. These new vertices should then be normalized to be at the same distance from the y axis as the others. The new vertices, along with pairs of the other four vertices, define 2 new quadrilaterals as shown below.



If the recursive counting variable is greater than zero, these two new quads should be divided (by calling divide_quad with the new sets of vertices). This is then repeated for as many levels as defined by the counter. (I recommend trying small numbers, like 0, 1 and 2 levels, to test it out). When the counter reaches zero, render the current quadrilateral from the four vertex parameters (by calling the quad( ) function).



You will need to specify a unique color for each face of your cylinder. I recommend alternating the color between two primary colors such as red and blue. You can do this by passing the color as a parameter to the divide_quad( ) function. Pass red to the first divide_quad() call and blue to the second call to divide_quad(), and so on. When the number of levels reaches zero, divide_quad( ) should pass the current color on to the quad( ) function.

You can use some of the following code to assist you with part (c) of the problem:

```
// Assuming you have a WebGL context named 'gl' and a shader program set up
```

```
// Define the initial vertices of the box
var vertices = [
    // Define your vertices here
];
// Define a recursive function to divide quadrilaterals
function divide_quad(v1, v2, v3, v4, count) {
    if (count == 0) {
        // Render the current quadrilateral
        quad(v1, v2, v3, v4);
    } else {
        // Calculate new vertices at the center of top and bottom edges
        var mid1 = [(v1[0] + v2[0]) / 2, (v1[1] + v2[1]) / 2, (v1[2] + v2[2]) / 2];
        var mid2 = [(v3[0] + v4[0]) / 2, (v3[1] + v4[1]) / 2, (v3[2] + v4[2]) / 2];
        // Normalize the new vertices
        normalize(mid1);
        normalize(mid2);
        // Recursive calls for the newly created quads
        divide_quad(v1, mid1, mid2, v4, count - 1);
        divide_quad(mid1, v2, v3, mid2, count - 1);
    }
}
// Define a function to normalize vertices
function normalize(vertex) {
    var len = Math.sqrt(vertex[0] * vertex[0] + vertex[2] * vertex[2]);
    vertex[0] *= 1 / len;
```

```
    vertex[2] *= 1 / len;
}
// Define a function to render a quadrilateral
function quad(v1, v2, v3, v4) {
    // Assuming you have a function 'drawQuad' to render the quad
    drawQuad(gl, v1, v2, v3, v4);
}
// Call the divide_quad function with your initial vertices and desired levels of subdivision
// Example: 2 levels of subdivision
divide_quad(vertices[0], vertices[1], vertices[2], vertices[3], 2);
```

What to submit for Problem # 02?

1. spin.html
2. spin.js
3. spin.png (image screenshot of your 3D box with UI elements visible)
4. cylinder.html
5. cylinder.js
6. cylinder.png (image screenshot of your 3D cylinder)