

## Question 1

a)

**Uniform** A global variable that cannot be changed by the shaders, and is dependent upon the geometric primitive in use.

**Attribute** A global variable that cannot be changed by the shader, which is dependent upon the current vertex, and therefore is only available within the vertex shader.

**Varying** A variable that stores data that has been interpolated, so can vary pixel by pixel. The vertex shader is able to modify this, however the fragment shader is not.

b)

```
1 var polygonVerticesAndColours = new Float32Array([
2     0.0,  1.0,  -4.0,  0.4,  1.0,  0.5,
3     -0.5, -1.0,  -4.0,  0.4,  0.7,  0.4,
4     0.5, -1.0,  -4.0,  0.5,  0.4,  0.5,
5     0.0,  1.0,  -2.0,  1.0,  1.0,  0.4,
6     -0.5, -1.0,  -2.0,  0.3,  1.0,  0.5,
7     0.5, -1.0,  -2.0,  1.0,  0.4,  0.4
8 ]);
9 var n = 6; // 6 vertices in this polygon
10
11 // Create a buffer object
12 var vertexColorbuffer = gl.createBuffer();
13 if (!vertexColorbuffer) {
14     console.log('Failed to create the buffer object');
15     return -1;
16 }
17
18 // Write the vertex information and enable it
19 gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorbuffer);
20 gl.bufferData(gl.ARRAY_BUFFER, polygonVerticesAndColours, gl.
    STATIC_DRAW);
21
22 var FSIZE = verticesColors.BYTES_PER_ELEMENT;
23
24 // Assign the buffer object to a_Position and enable the
    assignment
25 var a_Position = gl.getAttribLocation(gl.program, 'a_Position
    ');
26 if(a_Position < 0) {
```

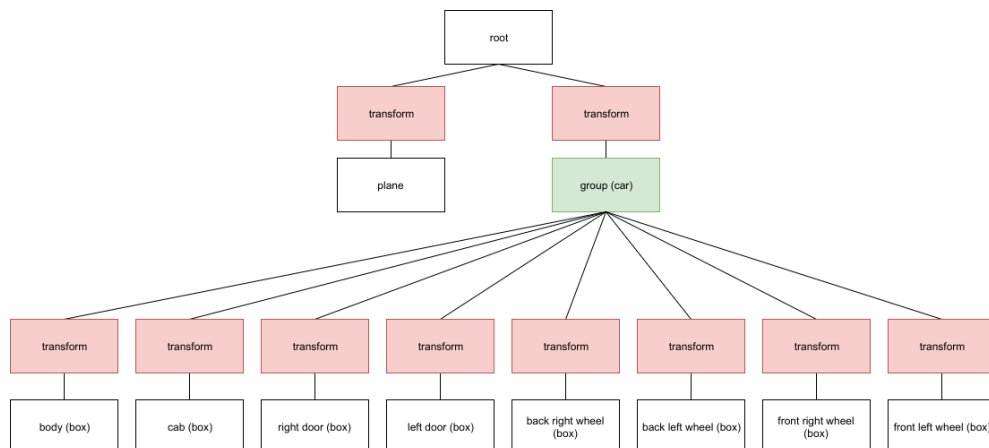
```

27 console.log('Failed to get the storage location of
    a_Position');
28 return -1;
29 }
30
31 gl.vertexAttribPointer(a_Position, 3, gl.FLOAT, false, FSIZE
    * 6, 0);
32 gl.enableVertexAttribArray(a_Position);
33
34 // Assign the buffer object to a_Color and enable the
    assignment
35 var a_Color = gl.getAttribLocation(gl.program, 'a_Color');
36 if(a_Color < 0) {
37     console.log('Failed to get the storage location of a_Color'
        );
38     return -1;
39 }
40 gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE *
    6, FSIZE * 3);
41 gl.enableVertexAttribArray(a_Color);

```

Listing 1: Colour and vertices in same array

c)



d)

i.

```
1 m.setRotate(angle, 0.0, 1.0, 0.0);  
2 m.translate(1.0, 3.0, -5.0);  
3 drawBox(m);
```

Listing 2: Explaining the meaning

The above code block ensures that the matrix  $m$  is rotated around the y axis by *angle* degrees from 0, regardless of current rotation before this code block, and then is translated by +1.0 in the x axis, +3.0 in the y axis, -5.0 in the z axis, relative to where  $m$  was before this code block. The matrix is then drawn.

ii.

In this case, replacing *setRotate* by *rotate* will *not* give the same result. *setRotate* is done before 'chain-able' transformations, however *rotate* is not, and so with *rotate* it would be 'chained' with *transform*, which would mean that the transform is actually carried out first, as the 'chain-able' commands are carried out in reverse order.

In this case, the matrix would be transformed away from its current position, and then rotated, still about the origin. If we assume it started centered about (0,0,0), it would be moved to (1.0, 3.0, -5.0), and then rotated about the origin, around the y axis, causing it to move in a big circle, as opposed to turning 'on the spot'.

## Question 2

The included files `tqvj24.html` and `tqvj24.js` are my code for this question.