L2 SM – Computer Graphics Summative Assignment

Questions

1a.)
Attributes take inputs as variables used in the vertex shader, for example, the vertex coordinates. These variables are dynamic meaning can be changed during runtime. Attributes are also global variables that may change per vertex, that are passed from the OpenGL application to vertex shaders.

Whereas **Uniforms** take inputs as variables available for the vertex shader and the fragment shader, for example, the light position. These variables are static meaning cannot be changed. Uniforms are also global variables that may change per primitive, that are passed from the OpenGL application to the shaders.

Varyings are used for interpolated data between a vertex shader and a fragment shader. They are also available for writing in the vertex shader, and read-only in a fragment shader.

Reference: http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/data-types-and-variables/

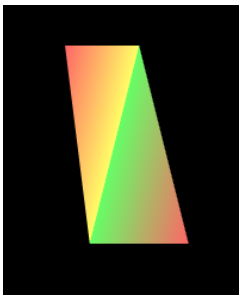
1b.)

Construction of an array containing 6 vertices where each is of the format (x,y,z,r,g,b) and the polygon produced:

```
var verticesColors = new Float32Array([
  // 2 triangles from 6 vertices
  0.75,  1.0,  -4.0,  0.4,  1.0,  0.4, // A green triangle
  0.25, -1.0,  -4.0,  0.4,  1.0,  0.4,
  1.25, -1.0,  -4.0,  1.0,  0.4,  0.4,

  0.75,  1.0,  -4.0,  1.0,  1.0,  0.4, // A yellow triangle
  0.25, -1.0,  -4.0,  1.0,  1.0,  0.4,
  0.00,  1.0,  -4.0,  1.0,  0.4,  0.4,
]);
var n = 6; // Three vertices per triangle * 6
```

:

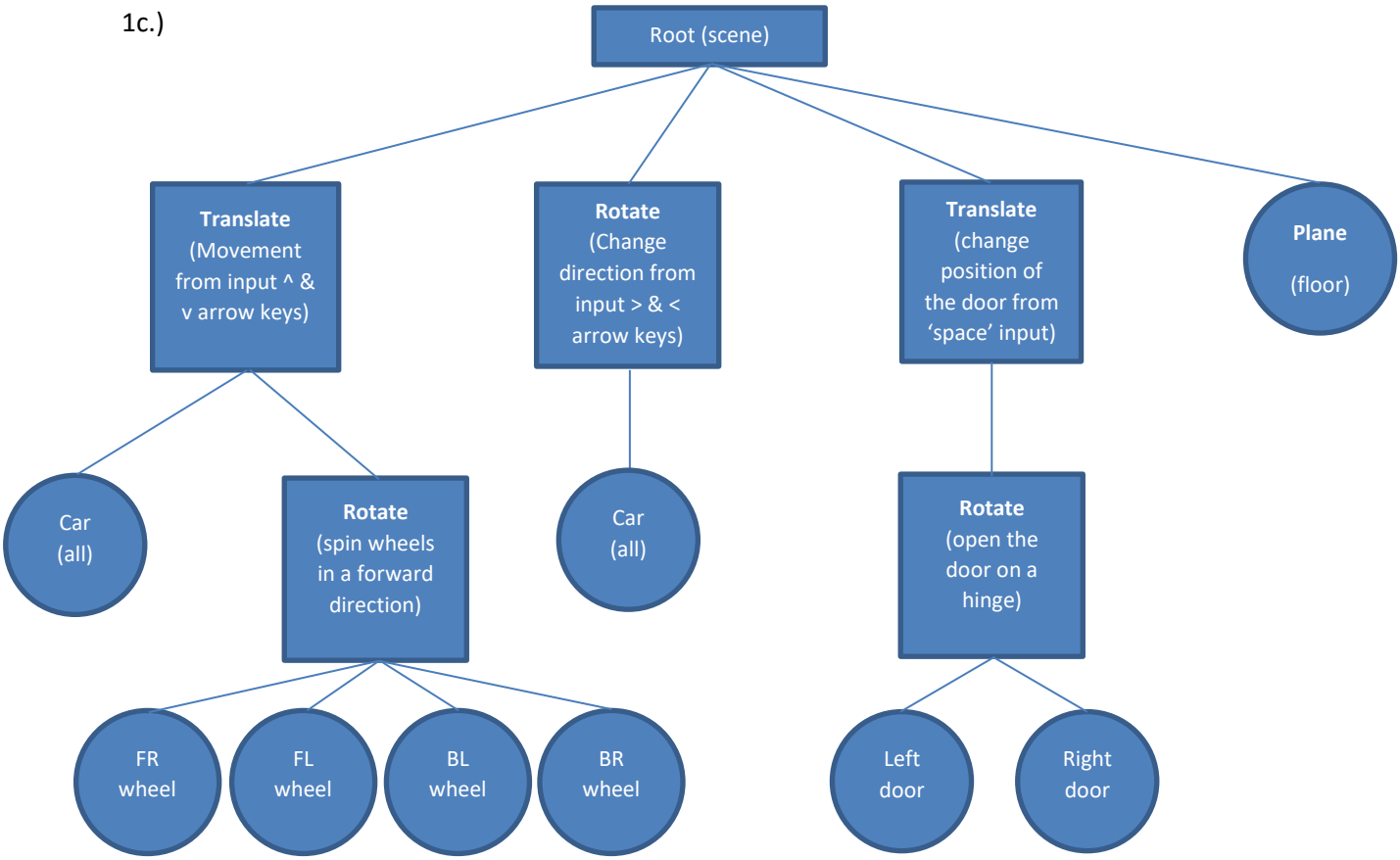


Constructing the corresponding vertex buffer objects and Assigning array data to become the position and colour attributes of the vertex shader:

```
  // Create a buffer object
  var vertexColorbuffer = gl.createBuffer();
  var FSIZE = verticesColors.BYTES_PER_ELEMENT;
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  var a_Color = gl.getAttribLocation(gl.program, 'a_Color');
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorbuffer);
  gl.bufferData(gl.ARRAY_BUFFER, verticesColors, gl.STATIC_DRAW);
  gl.vertexAttribPointer(a_Position, 3, gl.FLOAT, false, FSIZE * 6, 0);
  gl.enableVertexAttribArray(a_Position);
  gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE * 6, FSIZE * 3);
  gl.enableVertexAttribArray(a_Color);
```

Pklm51

1c.)



Car (all) includes: Body, roof, front right wheel, front left wheel, back right wheel, back left wheel, right door, left door. Each part also includes scaling and rotating to get it in the correct starting position.

1d.) i.)

Meaning: First, the code **draws** a box from the rotation matrix m. It then **translates** the box, 1 step in the positive x-direction, 3 steps in the positive y-direction and 1 step in the negative z-direction. The transformation matrix m is then **rotated** variable 'angle' degrees about the y-axis.

Results obtained: The result from this code is the drawing of a box which has been translated away and up from the centre, and rotated as follows:

Pklm51

1d.) ii.)

Yes, the result will be the same.

Seen as the object has already been translated, m.setRotate() would manipulate m in the same way as m.rotate() rotating the object about the y-axis. If previous manipulations had already occurred on the transformation matrix m, then the two would have different effects due to the reasoning below.

m.setRotate() **sets** the value of m to be the rotation matrix, rotating ‘angle’ degrees about the y axis, whereas m.rotate() **multiplies** the matrix stored in m by the rotation matrix, rotating ‘angle’ degrees about the y axis.