Interactive Dog

1. Create a hierarchical model of a (simplified) dog, composed of the following parts;
    a. Body
    b. 4 legs, each one composed of 2 independent components (upper and lower leg)
    c. Head
    d. Tail

    All components are cubes, use the cube function present in the file

To represent a complex figure, like a humanoid or a dog, it is possible to figure it with a hierarchical tree of rectangles as shown in the following figure, taken from Chapter 9 of book Interactive Computer Graphics: A Top-Down Approach with WebGL by Edward Angel and Dave Shreiner.

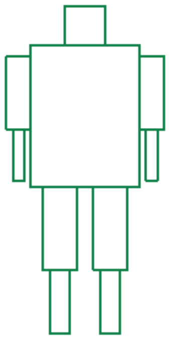

Figure 1: A humanoid figure

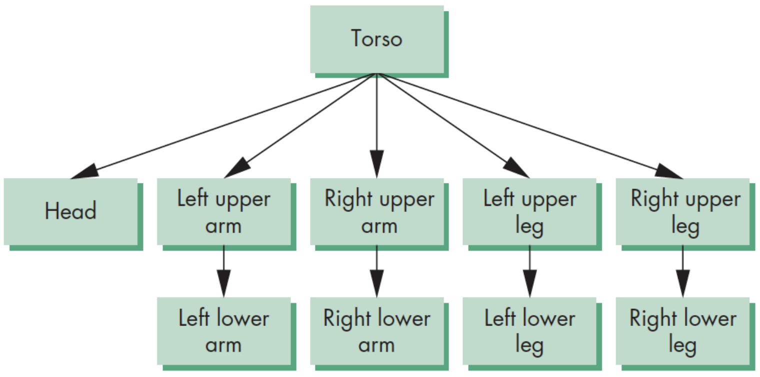

Figure 2: Tree representation

In .html:
It is only sufficient to remove the sliders.

In .js:
Remove document.getElementById.

To create the dog's tail I introduce variables and increase the number of nodes at 11. Theta must have same dimension of nodes number. Modifying width and height variables it is possible to make proper dog's dimension.
We have to add the tail to initiNodes function: a case of the switch can represent the tail and another appropriate function tail() can draw the tail.
Finally, with Theta angles it is possible to rotate the rectangles to make them look like a dog and with translate in initNodes it is possible to translate rectangles to attach them to the body.

2. **Add a procedural texture to the body of the dog. The texture should be a checkerboard pattern but with a linear decrease of intensity from the front to the back of the body. Use, as a reference, textureCube4 of Chapter 7 of the examples of the textbook. Notice however that you should not apply a sinusoid but a linear decrease in the direction of the tail.**

In .html:
In the vertex-shader script, I have to add varying vec4 fColor and varying vec2 fTextCoord and, in the main, fTextCoord = vTextCoord, so it is possible to program the next part in fragment-shader, where I have to add a variable varying vec2 fTexCoord, two uniform sampler2D (Text0 and Text1) and, in the main, I have to replace (or multiply) the fragment color vector with two texture2D functions that takes two parameters: sempler2D Text0/Text1 and varying vec2 fTexCoord.

In .js:
To have a procedural texture applied to the body of the dog, I add variables to manage texture coordinates and texture images: note that the image2 variable, which in the example of Chapter 7 had a sinusoid, now has a linear dicrease. Then it is needed to write the code of configureTexture function to configure as a texture the just added texture images variables. In init function I create a buffer with createBuffer function.

For the texture coordinates array, and than I program the vTextCoord variable for the vertex shader, I call configureTexture function previously created, and finally I pass to fragment shader the just created textures.

3. **Add a button that starts an animation of the dog so that, starting from an initial position where it is standing and positioned along the x axis, it walks to the right by moving (alternatively back and forth) the legs and turns the head in the direction of the viewer.**

In .html:
It's sufficient to add a button.

In .js:

I add a document.getElementById and a variable both linked to the HTML button to start the animation on the button click.

```
document.getElementById("startAnimation").onclick = function(event) {
         animate = !animate;
        translateBody = 0;
        rotateTail = 0;
        theta[headId] = 0;
        rotateLegs[0] = 0;
        rotateLegs[1] = 0;
        rotateLegs[2] = 0;
        rotateLegs[3] = 0;
        rotateLegValue = 1.0;
        rotateLegValue2 = -1.0;
        rotateLowerLegValue = 2.0;
        rotateLowerLegValue2 = 2.0;
};
```

By clicking again the button all the variables come to a default value, making the dog return to the initial position.
Now it is needed to move the initNodes function call inside the render function.
Finally, inside the animation function, an if-else structure does the controls to check if it is the case to switch the rotation of dog's components or not (e.g. if the head is rotated less than 90 degrees, it keeps rotating a bit more until it reaches this value and than the head stops rotating; the tail rotates until 45 degrees, than the value becomes negative so that it restarts the animation).
For example, it follows the code for head's animation

```
// Head animation
    if (theta[headId] < 90) theta[headId] += 2.0;
```

When animate button is clicked it is needed to restart all the variables to reset the initial situation.