Homework 2 - Interactive Graphics

Santilli Sofia - 1813509

May 2021

1 Exercise 1

Create a hierarchical model of a (simplified) sheep https://en.wikipedia.org/wiki/Sheep, 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. The sheep has a white/light grey color.

The sheep is constructed through a hierarchical structure. This structure consists in a tree: in the code each node can have a child and a sibling (brother). The root is the *torso* of the sheep, that has the *head* as child. The *head* has the left upper arm as sibling, etc. The entire structure is represented in the *Figure 1* below.
Starting from the given structure of the humanoid, I simply did rotations and translations and I changed the heights and widths of the different parts of the body, in order to obtain the desired shape of a sheep. Finally I added a node to the tree (also this component is a cube) to be the *tail* of my sheep and I set for it a rotation of -70° (-70 has been specified in the *theta* variable) around the z axis, in order to have the tail inclined. The tail is the sibling of the *right upper leg*, has no child and no sibling. In order to add the tail I also increased the *numNodes* and *numAngles* variables. In order to do the animation (we'll discuss later) also the rotation of the components for the upper and lower legs has been set to happen around the z axis.
This hierarchical structure has a practical application in animation. In fact, for example, in order to translate the whole sheep, it will be sufficient to translate the *torso* of the sheep: all the other components of the body (as they are children of the torso or child of a child of the torso) will follow it.
In order to set the grey color of the sheep (successively removed to add textures) I simply changed the color passed in the fragment shader *fColor = vec4(0.8, 0.8, 0.8, 1.0);*.

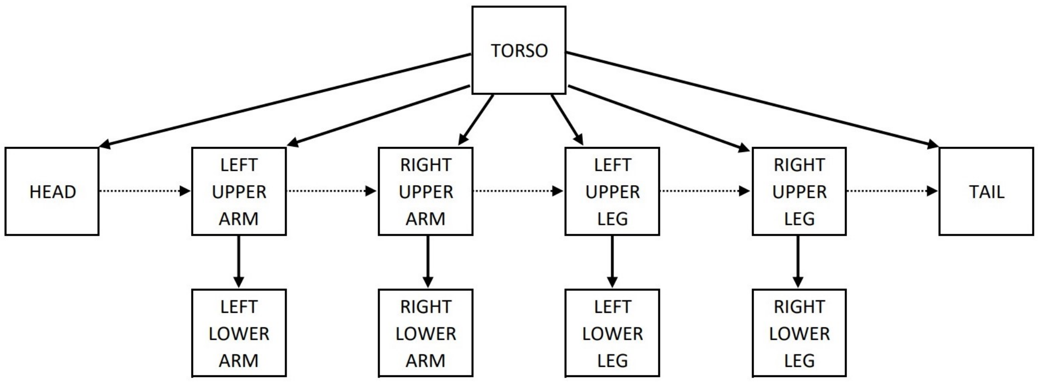

Figure 1: Hierarchical model of the sheep. The dotted horizontal arrows point out how siblings are connected in the code.

2 Exercise 2

Add a surface on which you position the sheep that corresponds to a grass field. Attach to it a texture (color, bump or both) to give the appearance of a grass field.

In order to add a grass field, I created a new node, which is independent from the sheep model (and we'll see also from the fence). In order to draw both the sheep and the grass field on the canvas, in the *render* function, the *traverse* function is called two times: the first time passing as argument the *torsoId* (that draws the torso and all the other components of the sheep connected to it); the second time passing as argument the *grassId* (that draws only the cube for the grass, as it is a tree made up of a single node).
To give an appearance of a grass field, I loaded an image *'grass2.jpg'* and I passed it as an argument in the *Configure-Texture* function. The strategies applied in order to handle different textures for different components will be better explained in the next paragraph.
For the height of the grass cube has been chosen a large value (12.0): in this way the texture attached to the lateral faces doesn't look too deformed.

3 Exercise 3

Load or generate at least two more textures. A color texture to be attached to the front face of the head and a bump texture to be applied to the sides of the body to give the "wool effect".

I added five different textures, listed in the following table (where it is present also the color assigned to the lower legs of the sheep).

object in the scene	***index***	***texture/color***
grass	1.0	*grass2.jpg* image texture
fence	2.0	*wood.jpg* image texture
head, upper legs, tail (sheep)	0.0	*wool.jpg* image texture
face (one face of the cube of the head)	0.3	*wool_face_.jpg* image texture
body/torso of the sheep	0.2	bump texture
lower legs (sheep)	0.1	light pink color

Table 1: Textures and colors used for the different components of the scene.

In the js file, the function *ConfigureTexture(image1, image2, image3, image4, image5)* initializes the different textures. In fact, in the *init()*, all the five different textures used are passed as argument to the *ConfigureTexture()* function. In order to make it work, it is necessary to have, for each texture, the following lines in the *ConfigureTexture*:

```
    texture_wood = gl.createTexture();
    gl.activeTexture(gl.TEXTURE1);
    gl.bindTexture(gl.TEXTURE_2D, texture_wood);
    ....
    gl.uniform1i(gl.getUniformLocation(program, "uTextureMap_wood"), 1);
```

In order to make the vertex and fragment shader distinguish between the different texture/colors to handle depending on the component of the scene, I decided to pass a uniform variable (a float) *uObjectIndex* to the shaders. This variable is passed from the js file in the functions: grass(), head(), torso(), leftUpperLeg(), fenceLateral1(), ... The indices related to the different objects (or parts of the objects) in the scene are shown in *Table 1.*

In order to use a different texture on the face of the head, I simply had to call more times the *drawArrays()* function, each time passing the right uniform variable (index) to the shaders.

For the Bump texture, I created manually a texture which gives a sort of "wool effect", based on an array of zeros and ones. In order to simulate small displacements on the surface, normals and tangents to the faces of the cubes have been used. As a tangent to a face, I decided to use a side of a face *t1*. A *normal buffer* and a *tangent buffer* were used. The shaders handle the light in order to give a bump effect to the texture on the solid and also to light up the other objects in the scene. In fact, in order to realize the bump texture on the torso of the sheep, it has been necessary to add a light in the scene, positioned along the positive z axis, at the right-frontal side of the sheep. Then, in order to light up also the other objects in the scene, I added also the material properties. The presence of only one light in the scene makes the back of the scene darker (recognizable by moving the camera).

4 Exercise 4

Create a (very simplified) model of a fence and position it on the surface and near the sheep.

Also the fence has been created as a hierarchical model: the *fenceLateral1* is the root and has *fenceLateral2* as child. The last one has the *fenceHorizontal1* as sibling. Finally *fenceHorizontal1* has *fenceHorizontal2* as sibling.
The model of the fence is indipendent from the grass and the sheep. Therefore, in order to draw it on the canvas, it is necessary to call (in the *render* function) the *traverse* function again, this time passing to it the *fenceLateral1Id*.
Similarly to what we've done for the grass and the textures of the sheep, also in order to apply a wood texture to the fence, an uniform variable of value 2.0 has been passed to the shaders and an image *'wood.jpg'* has been loaded and passed to the *ConfigureTexture* function as second argument.

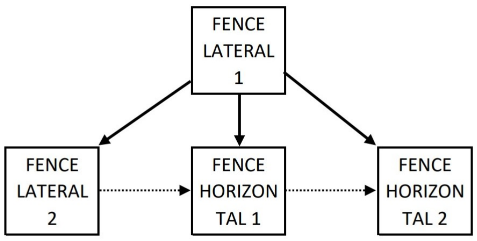

Figure 2: Hierarchical model of the fence. The dotted horizontal arrows point out how siblings are connected in the code.

5 Exercise 5

Add a button that starts an animation of the sheep so that, starting from an initial position where it is in a walking mode, it walks on the surface towards the fence by moving (alternatively back and forth) the legs, then jumps over the fence and lands on the surface on the other side of the fence.

I added two buttons: a *Start button* and a *Reset button.*
When the*Start button* is clicked, the animation starts: through the *setInterval()* method, every 120 milliseconds, the function *animate()* is called. In this function, I split the animation in different steps, depending on the point on the x axis the torso of the sheep is:

- (x<7) the walking is obtained by translating the torso of the sheep (and automatically also the other connected components of the sheep will translate), by calling the function UpdateNodes(). This function has been created only in order to update the position of the torso in the xy plane, by passing to the function the values *inc_x*, *inc_y*: these values express how much we want to translate the object along the x or the y direction.
  Simultaneously with the translation, the upper and lower legs of the sheep are rotated around the z axis, by updating the correspondent values of the angles in the *theta* variable and calling the *initNodes()* with the correspondent index.
  The walking itself is structured in 3 steps, activated through two flags: *switchLegsMoved* and *transitionLegs.* The first flag points which pair of legs have to move back and forth. The second flag is True when all the upper and lower legs are characterised by their starting orientation (in order to make more visible and realistic the animation).
- (x<7.5) the sheep moves up the frontal part of the body (by rotating the torso of -45° around the z axis), getting ready to jump. At the same time, also the upper and lower legs rotates a bit around the z axis.
- (x<11.75) the first phase of the jump (the elevation): we keep rotating and translating (both x and y increase) the torso and rotating the legs.
- (x<18.5) the second phase of the jump (the descent): the torso keeps rotating and translating (x increases, y decreases) and the legs rotate.

- (x=18.5) the landing on the grass: the orientations of all the angles of rotation of the components of the sheep are posed to the initial value of *theta= [0, 0, 180, 0, 180, 0, 180, 0, 180, 0, -70, 180, 0, 0]; .*

The animation during the jump appears slower with respect to the walking; it is due to the fact that the jump phase has been constructed with a bigger number of frames and smaller translations.

When the *Reset button* is clicked, it is called the instruction *window.location.reload()*, that reloads the current document, so both the sheep and the position of the camera return to their initial position.

6 Exercise 6

Allow the user to move the camera before and during the animation.

In order to move the camera I introduced the parameters used to compute the modelView matrix (*eye* (so *thetaCamera* and *phiCamera*), *at*, *up*) and the parameters for the orthogonal projection (*left, right, bottom, top, near, far*). The projection matrix and the modelView matrix are initialized in the *init()* function and passed to the shaders; then, in order to have the possibility to move the camera before and during the animation, in the *render()* function they are computed again and passed to the shaders. As the normal matrix depends on the modelView matrix, also this matrix is computed and passed to the shaders both in the *init()* and in the *render()*.
Buttons and sliders have been added in order to allow the user to move the camera. In particular we can see how, the scene initially has *near* = -23.0 and *far* = 23.0. By increasing this range nothing changes. But if we decrease this range gradually, parts of the scene disappear (first the grass, then the sheep and finally the fence).

Pay attention to the fact that, moving the camera, the light position remains constant in the cartesian space[1]: the light remains always on the left-frontal side of the sheep, a bit elevated along the y axis. Therefore it is normal that, moving the camera, the back of the sheep (and of the rest of the scene) is always in shadow.

[1]In order to do this, in the vertex shader, I multiplied the uniform variable *uLightPosition* for the *modelViewMatrix*, both in case of bump or normal surfaces:
*vec3 eyeLightPos = (modelViewMatrix*uLightPosition).xyz;*
*vec3 light = (modelViewMatrix * uLightPosition).xyz;*