Interactive Graphics - Homework 2

Michela Proietti - 1739846

June 6, 2021

Step 1

The figure below shows the hierarchical model of the sheep. As we can see, the root of the tree is represented by the sheep's body. This means that the positions of all the other parts of the sheep will be defined with respect to its body, so if we rotate or translate the body, the other parts will follow its movements. We associated one index to each part of the sheep, while the head has two indices, since we wanted to be able to control the rotation both around the x and z axes. In fact, these indices are used to access the *theta* array, that contains the angles by which we want to rotate each component. Initially, they are all set to 0, exception made for the one that controls the head's rotation around the z axis, since the head is a little inclined, but they are then changed in order to animate the sheep. Another important aspect is that the lower parts of the sheep's legs are all children of the correspondent upper parts. In this way, the lower parts will follow the upper parts of the legs during the animation. Finally, we have modified the *initNodes* function and the functions to render the various parts of the sheep, also adding the function *tail* that was not present, to correctly position and render all the components. Initially, in order to correctly color the sheep we have simply set the color of each fragment to light gray in the fragment shader.

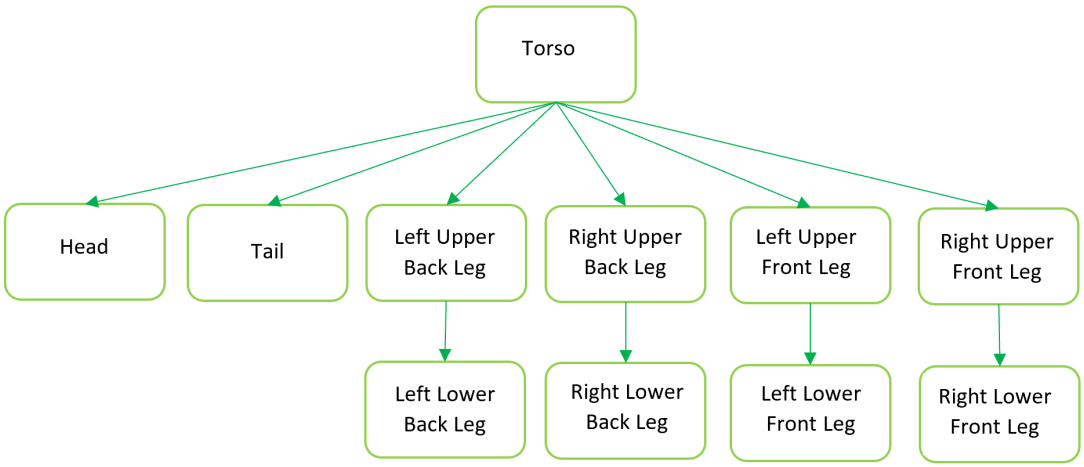

Figure 1: Hierarchical model of the sheep

Step 2

There are various methods that have been tested in order to create the grass field, but in all cases it has been modelled as a parallelepiped. The first thing that has been tried was to add it as a completely separated object. In order to do this, we have created a new mat4 named g in the *initNodes* function and we modelled the grass separately from the sheep using this matrix. Then, we have created the *grass* function and we have added *traverse(grassId)* in the render function to display it on the screen. However, since the two objects belong to the same scene, we tried to add the grass to the tree structure used to model the sheep, and there are two ways to do this. One possibility is to add the grass as a sibling of the sheep's body, thus having two independent subtrees. The second possibility is to make the grass the root of the tree, having the sheep's body as child. The three methods that have been presented work exactly in the same way in this simple application, but we decided to adopt the last method, because if for example we wanted to rotate the grass, it would be more correct to make the sheep rotate with it, since it is placed on it. To give the appearance of a grass field, we have loaded a color texture that has been applied to all the faces of the parallelepiped.

Step 3

The color texture that has been applied to the front face of the sheep's head is actually very simple, because it just draws a light pink rectangle which represents the mouth of the sheep. To better give the appearance of a face, we have then attached a color texture to the front and back face of the cube representing the head that simply draws a black rectangle representing an eye. In order to apply a different texture to the various faces of the cube, in the *head* function we have used a uniform variable that is set to 1 when we want to use the texture for the eyes, to 2 if we want to apply the texture for the mouth, and to 0 otherwise every time before calling the *drawArrays* function. Another possible way of representing the face is to draw both eyes and mouth on the front face of the head and use the bump texture on all other faces, but we decided to use the first alternative because it seemed more realistic. The figures below show the two different representations. The bump texture for the sheep's body, that has been also applied to the upper parts of the legs, to the tail, and to the top and left faces of the sheep's head, has been defined using random numbers. In particular, we loop over the size of the texture and at each iteration we generate a random number and we set the data value to 0.7 (to lower the contrast in order not to have a black and white sheep) if the random number is even or 1 otherwise. In order to render the bump texture, we added a point light source in the scene, and we have added an if clause to handle the rendering of the bump map in the shaders. To be more specific, we have used an integer uniform variable that is set to a different value depending on the object we are currently rendering and it is sent to the shaders inside the functions used to render the various components. In this way, we can use its value to understand how to do each rendering, which texture to apply, how to handle light and so on. Since we have added a light to correctly render the bump texture, we have then modelled the light for all the objects, implementing the Phong shading model. However, the scene was still very dark, and whatever the position of the light was, half of the sheep was completely in the shade. Therefore, we decided to add a second light source, to have a more uniform lighting of the scene to better show all its features.

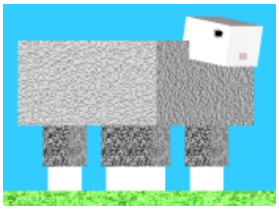

Figure 2: Textures that has been used for the sheep's face

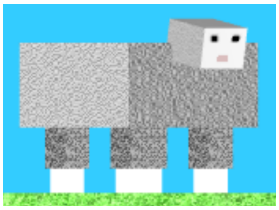

Figure 3: Alternative texture

Step 4

The fence is made up by three parallelepipeds, two of them placed vertically, and the other one horizontally. The same reasoning that has been done while adding the grass field holds also in this case, so we made all the blocks of the fence siblings of the sheep's body, and therefore children of the grass node. In this way, if we rotate or translate the grass, everything that is on it will move with it. Since the three blocks of the fence have been modelled as siblings, they can be moved independently one from the other. Even in this case, we have used a color texture to give the fence the appearance of wood. All the textures are configured inside the *configureTexture* function, which is called in the *init* function. Initially, each texture was configured and sent to the shaders inside the function of the component that we were rendering in that moment. However, this slowed down the animation and introduced some lags, because there was a continuous exchange of information between the CPU and the GPU. By configuring and sending to the shaders all the textures together, we managed to solve this problem.

Step 5

The animation is handled by the *start* function, that is called inside the handler of the button that starts the animation itself. Initially, the sheep is in a walking mode until it reaches a certain position in front of the fence. In order to walk, we increase or decrease the angle of the upper and lower parts of the legs, moving synchronously the front and back opposite legs (right front / left back and left front / right back). Clearly, since the lower parts of the legs are children of the upper parts, we rotate each lower part in the opposite direction with respect to the corresponding upper part, so that the sole of each foot remains parallel to the ground. At the same time, we need to increase the x coordinate of the sheep's body itself, and since it is the parent of all the other parts, the legs, the head and the tail will translate with it. Once the sheep arrives in front of the fence, in order to make the jump more realistic, we make the sheep stand

on the back legs before detaching from the ground. To do this, we change the inclination of the sheep's body, and we also move the legs to have them well positioned. Then, while the sheep is jumping, the x and y coordinates of the sheep's body are increased and its inclination starts to be decreased until the sheep reaches a certain height. At this point, there is the landing phase, in which the body is tilted forward and the legs are brought back to their original position. When the sheep gets close to the ground, its body start to be brought back to its horizontal orientation. Finally, once the sheep returns in its original position, it goes back to the walking mode and it does few steps until it reaches the end of the canvas.

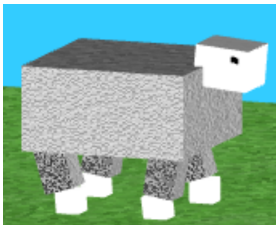

Figure 4: We can see how the front right and the back left legs move together, as well as the front left and the back right ones while the sheep walks. Moreover, we can notice that the feet remain always attached and parallel to the ground.

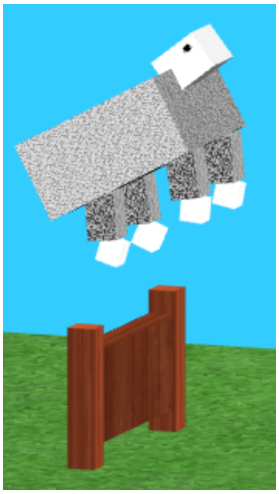

Figure 5: While the sheep is jumping, the front and back legs are parallel to each other.

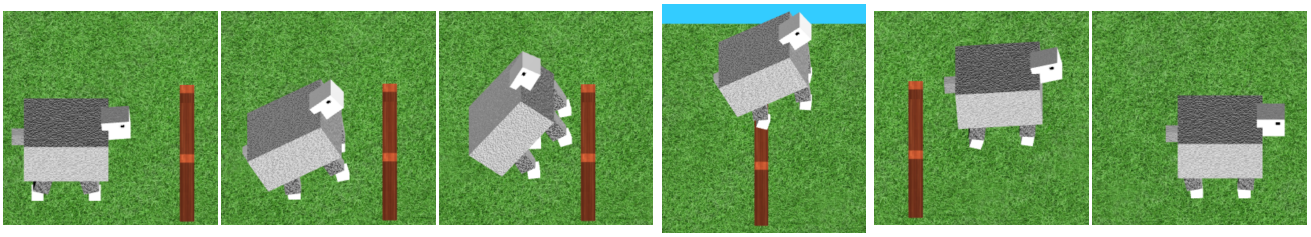

Figure 6: In this sequence we show how the lights interact with the objects. One light is on the left, very high and far along the z axis, thus lighting the frontal part of the sheep and partially also the top and left part. We can see that when the sheep stands to jump, the top part of the body and the head become more lit because of this first light, and then becomes darker again. The second light is placed on the right, behind the sheep, and it is not very high. Therefore, we can see that when the sheep stands, the right faces of the head and the body become darker, because the light is lower, and then they become lighter again.

Step 6

In order to move the camera, we have first defined the modelViewMatrix in the render function, using the initial values of the viewing parameters so that when we call the *initNodes* function we will be able to do the rendering, but then we recompute the viewer position and the modelViewMatrix even in the *render* function, so that we are able to take into account the changes done on the viewing parameters through some sliders. We can notice that in the orthographic projection if we change the radius, the size of the objects does not change, because it is a parallel projection and therefore we are infinitely far away from the objects. However, if we set the radius to its maximum value, we see that the objects disappear, because they go out of the viewing volume.