Interactive Graphics 2^{nd} homework

Fabrizio Casadei - 1952529

March-April 2021

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.

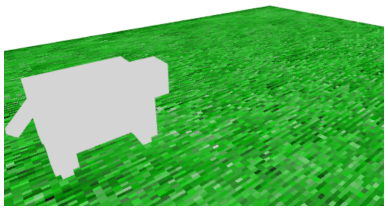
Starting from the hierarchical model already defined, has been modelled a simplified version of a sheep, using cubes. A function to create each node has been exploited, in which each part of the sheep is defined in terms of the transformation matrix, render function, sibling node and child node.
Modifying the transformation matrix and the properties like location and scale factor in the pre-existing model, almost all the components are been carried out.
A new element of the hierarchical model has been defined from scratch, and is the node that represents the tail, adding properties information like the id, the width and the height of the cube.
In the render function invoking the creation starting from the torso id, recursively the complete sheep is generated.
Below it's shown the result.

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.

To create the grass has been realized a specific render function in which is sent a flag to the fragment shader to apply a colour texture and is built the instance matrix multiplying the model view matrix with specific scale, rotation and translation matrices, in a way of having a correct representation of the grass.
Then a new node (with a new id) independent from the ones of the sheep (no connections in terms of sibling and child), has been built and called for the creation in the render function.

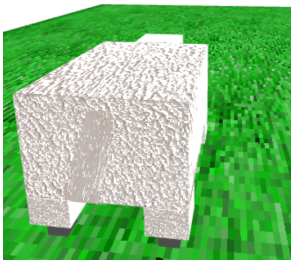
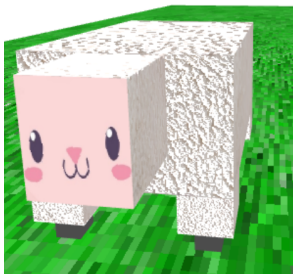
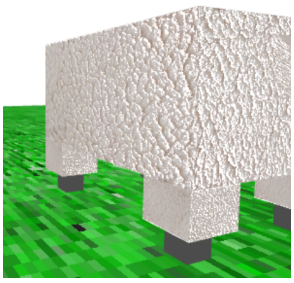
To load the color texture has been converted an image in an array of values which represents the RGB 8 bit values of the intensities, then the corresponding texture is generated and sent to the fragment shader for the application. This solution prevents the launch of a web server for loading the images (Cross-Origin Resource Sharing). The grass aspect is the following.



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".

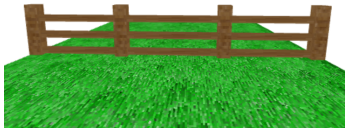
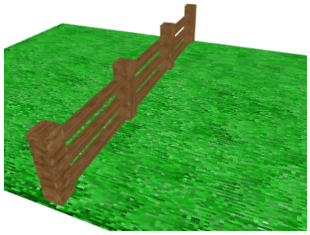
With the same strategy used for the previous exercise, hence, through the sending of flags to the fragment shader, we handle different cases to twist the aspect of the sheep.
the first step has been the loading of images and the generation of the relative textures. Images are managed as arrays of values like for the grass case (these values are been carried out using a script written in python), in the specific we used 3 different images: one is the wool RGB image, then we have the same in grey-scale to generate the bump texture, and the last one is an RGB image which models the face of the sheep. To display the result from the bump texture, has been defined a light point and computed all the Light calculus in the vertex shader, then in the fragment shader are computed normals according to the bump texture and used the light information to carry out the color.
In order of having a better and more realistic representation of the sheep aspect, the final color is computed adding to the bump texture term the corresponding color texture with a certain scale factor. The result it's shown below.

Exercise 4

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

In the same way, as for the grass has been implemented a render function that creates the fence.
Stretching the cubes, positioning, and rotating them has been built a kind of intersection of cubes which looks like an actual fence. A new independent node with its own id has been defined and called for the creation in the rendering function like for the grass and the sheep case.
Moreover, with the same manner used for the other textures, has been handled the application of a wood look to the fence. As always in the below part are presented the results.

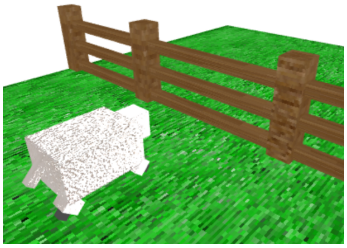
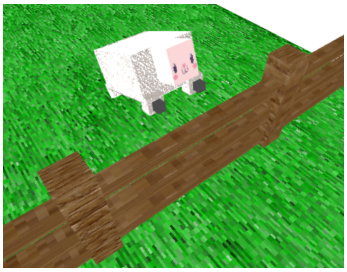
Exercise 5

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.

To build the animation, several functions for all the actions have been defined. Setting an interval in the javaScript file this list of function is evaluated. The specific next action for the animation change based on the position of the sheep's torso. In the specific the functions can be summed up with this list:

- the walk animation function: 4 different configurations of the arms/legs, changed sequentially to give the idea of a correct walk on the grass, and the same time the sheep is shifted forward.
- jumping animation increasing the height: the sheep still goes forward, the legs/arms are positioned for the jump and the sheep has been inclined correctly to emulate the jump.
- jumping animation with invariant height: the jump continue moving the sheep ahead without changing the angle between the grass and the sheep. This animation is activated approximatively when the sheep is above the fence.
- the landing on the grass after the jump: the sheep goes down to the grass decreasing the inclination and going forward.

Rotations and translations are applied to the torso because having a hierarchical model all the other parts of the body change according to it. After the landing, the animation of the walk is used again and then the sheep rotate to face the fence. The animation is controllable with buttons, starting and stopping the animation and resetting it.
Below are shown some captures from the animation.


Exercise 6

Allow the user to move the camera before and during the animation.
To compute this final task has been defined 3 apposite sliders that manage respectively:

- motion of the camera around the x-axis.
- motion of the camera around the y-axis.
- distance from the camera to the scene (radius)

from the sliders settings have been computed the x,y and z values for the eye and then passed to the function **lookAt** to compute the model view matrix, which is used in the vertex shader to compute the position of each vertex.
In this way, the user has complete control of the camera.