Homework 2 - Interactive Graphics

Leandro Maglianella - 1792507

I have tested my code using both Google Chrome and Firefox browsers.

1. Create a hierarchical model of a (simplified) sheep, composed of: body; 4 legs, each one composed of 2 independent components (upper and lower leg); head; tail. All components are cubes. The sheep has a white/light grey color.

The sheep is represented by a hierarchical structure. Hierarchical structures allow to represent relationships between very complex and articulated objects. In particular, they use a "tree" structure where the model's components are nodes, each one connected to a sibling and/or a child (if they have one). Hierarchical structures are particularly useful in the field of animation: they are in fact made so that every child node is subject to the same translations and rotations as its parent node (i.e., every child node belongs in the reference frame of its parent), instead of having to move every component singularly. In this case, I created the model of the sheep constructing it using the proposed parts and, also, I added two eyes: I preferred to do so instead of just drawing them in the head's texture to give a more 3D/realistic appearance to the sheep. The general model nodes' structure is as follows (arrow pointed down = child; arrow pointed right = sibling):

$$\begin{tikzcd}
\boxed{\text{Body}} \arrow[d] & & & & & & \\
\boxed{\text{Head}} \arrow[d] \arrow[rr] & & \boxed{\text{LeftFront UpLeg}} \arrow[r] \arrow[d] & \boxed{\text{RightFront UpLeg}} \arrow[r] \arrow[d] & \boxed{\text{LeftRear UpLeg}} \arrow[r] \arrow[d] & \boxed{\text{RightRear UpLeg}} \arrow[r] \arrow[d] & \boxed{\text{Tail}} \\
\boxed{\text{Eye}} \arrow[r] & \boxed{\text{Eye2}} & \boxed{\text{LeftFront LowLeg}} & \boxed{\text{RightFront LowLeg}} & \boxed{\text{LeftRear LowLeg}} & \boxed{\text{RightRear LowLeg}} &
\end{tikzcd}$$

I decided to structure my sheep in this way to replicate a real-life sheep: as said, in this way, if the body moves then everything moves; if the head moves then the eyes move; if an upper leg moves then the corresponding lower leg moves. Every component is created as a cube, each one that I have then translated, rotated and scaled to form the sheep. Every cube is (obviously) formed with 8 vertices and 6 quad faces: to each vertex is associated its 4D position, 3D normal, 2D texture coordinate and 3D tangent. The sheep had a light grey colour, I then modified this when I applied the textures.

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.

As for the sheep, I created the grass field as a hierarchical model consisting of only one node (just for now, I will then use the grass as the fence's parent node). It is just an extensive and very thin cube: I decided not to make it too extensive to be able to show it in its entirety in the viewing volume. I have attached to it a texture mapped from an external jpg image (here on the side):

Finally, I also modified the canvas background colour to sky blue to give an even more realistic appearance to the scene.

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 body to give the “wool effect”.

I have loaded another external jpg image and attached it to the sheep’s head: I applied the same texture to the lower legs as well. For the body (but also for the upper legs and the tail) I generated a bump texture using as bump data a 40×40 image of random values between 0 and 0.5 (I tried some values and I liked the “wool effect” generated by these parameters); the bump texture is then sent to the fragment shader where the normals are perturbed. During this phase, I added a light source and a material to compute all the components of the Phong shading model’s illumination equation which I used in the fragment shader for the whole scene. Finally, I applied a black colour to the eyes.

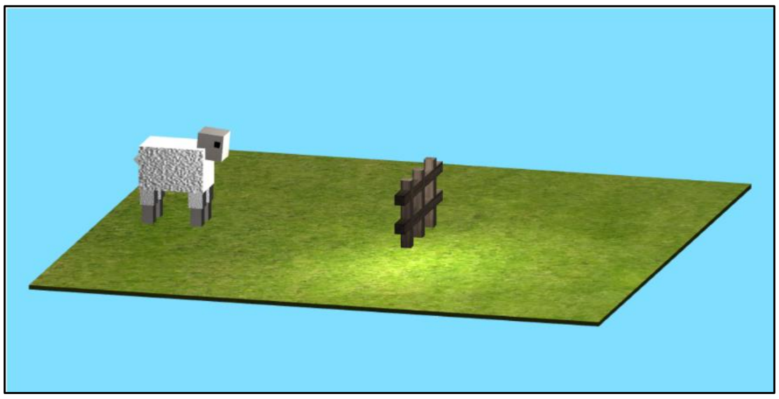
4. Create a (very simplified) model of a fence and position it on the surface and near the sheep.

As anticipated in point 2, I created the fence as child of the grass node in its hierarchical model: I did so just to ideally link the fence to the grass and to not create a whole new unnecessary model, however this was not essential since the grass is never subject to movement in my project. Moreover, I did not also make the sheep’s body a grass’s child so as not to complicate a single model too much. The fence’s shape was represented using 5 cubes jointed together. As before, also for the fence I loaded an external jpg image as texture:

Finally, the model’s structure is as follow:

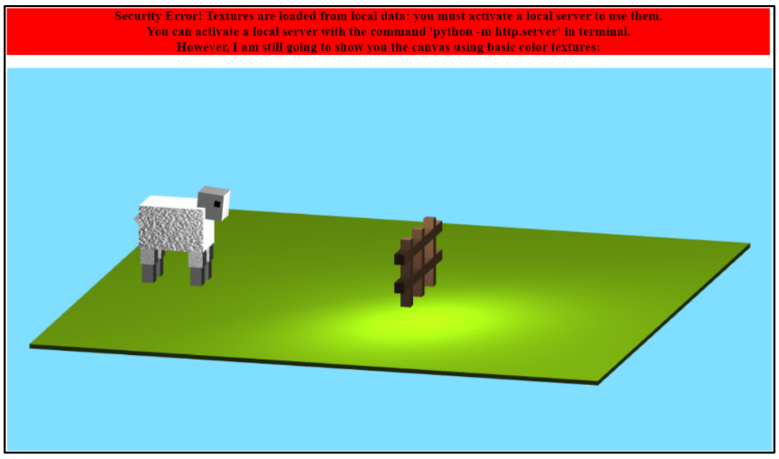
$$\begin{tikzcd}
\boxed{\text{Grass}} \arrow[d] & & & & \\
\boxed{\text{Fence1}} \arrow[r] & \boxed{\text{Fence2}} \arrow[r] & \boxed{\text{Fence3}} \arrow[r] & \boxed{\text{Fence4}} \arrow[r] & \boxed{\text{Fence5}}
\end{tikzcd}$$

The resulting canvas is shown here on the side:



At this point I just added some little extras to make my project more user-friendly: because most of the textures are loaded from local data, if the user fails to correctly pass the images to the application using a local server, I catch the error and then I show a message on screen with the loading instructions. Moreover, I still show the canvas using basic homogeneous colours (grey, green and brown) instead of the textures.

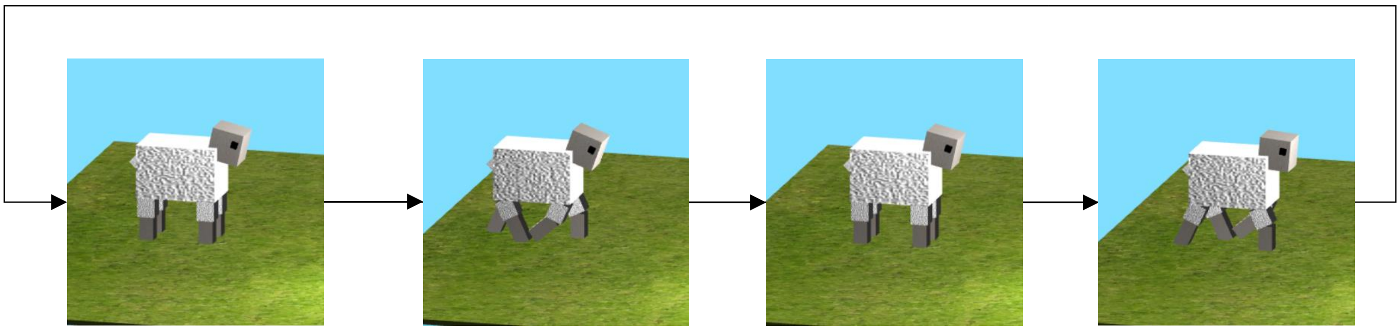
The resulting “error” canvas is shown here on the side:



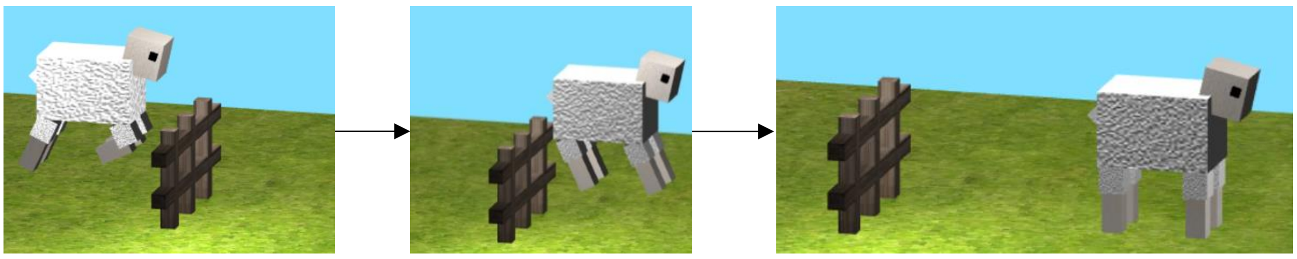
Security Error! Textures are loaded from local data: you must activate a local server to use them.
You can activate a local server with the command 'python -m http.server' in terminal.
However, I am still going to show you the canvas using basic color textures:

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 created two buttons: one to start the animation and one to reset the sheep to its initial position. The animation was created changing the positions and rotations of the sheep's components and re-initializing them (the animation function is called every 150ms using the setInterval function): when it starts, the sheep begins to move towards the fence in a walking mode incrementing its body *x* coordinate. Meanwhile, the upper and lower legs move back and forth varying their joint angles and the head oscillates to simulate a realistic walk. To summarize, as the sheep moves forward, its components cyclically take on these orientations:

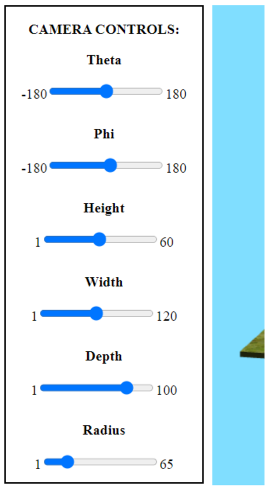


The velocity of the sheep is not constant, instead it accelerates linearly as it progresses. As the sheep gets closer and approaches the fence, it starts the jump animation's take off: during this phase the sheep increments its *x* and *y* coordinates; as soon as the fence is overtaken, the landing phase begins where the sheep continues to increase its *x* coordinate while decreasing the *y*. During the take off and the landing the sheep's legs assume realistic angles too. Finally, when the sheep lands, it goes back to walking mode, takes a few steps and stops:



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

The *modelViewMatrix* is computed with the *lookAt(eye, at, up)* function while the *projectionMatrix* is computed with *ortho(left, right, bottom, top, near, far)*: I have created six sliders with which the users can interact to move the camera whenever they want.



In particular:

- *Theta*, *Phi* and *Radius* control the camera eye's position.
- *Width*, *Height* and *Depth* control the dimension and the aspect of the viewing volume.
- *Width* controls the *left* and *right* parameters (left and right bounds of the frustum).
- *Height* controls the *bottom* and *top* parameters (bottom and top bounds of the frustum).
- *Depth* controls the *near* and *far* parameters (near and far bounds of the frustum).