Interactive Graphics - Homework 2

Marco Lo Pinto, 1746911

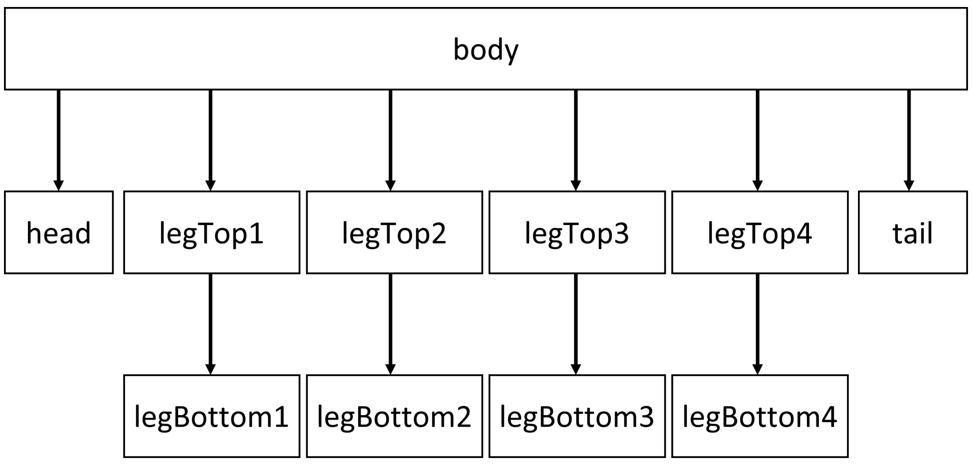
May 24, 2021

The homework uses Object Oriented Programming Syntax in JavaScript (ES6), so to have modularity and to reuse of code through inheritance. It also exploits promises with async/await syntax to load the images. This project has been tested both on Google Chrome (version 90) and Edge (version 90). It was also tested on Firefox (version 86). It is important to create a local server in order to bypass browser restrictions.
This homework uses the classes and the code from Homework1.

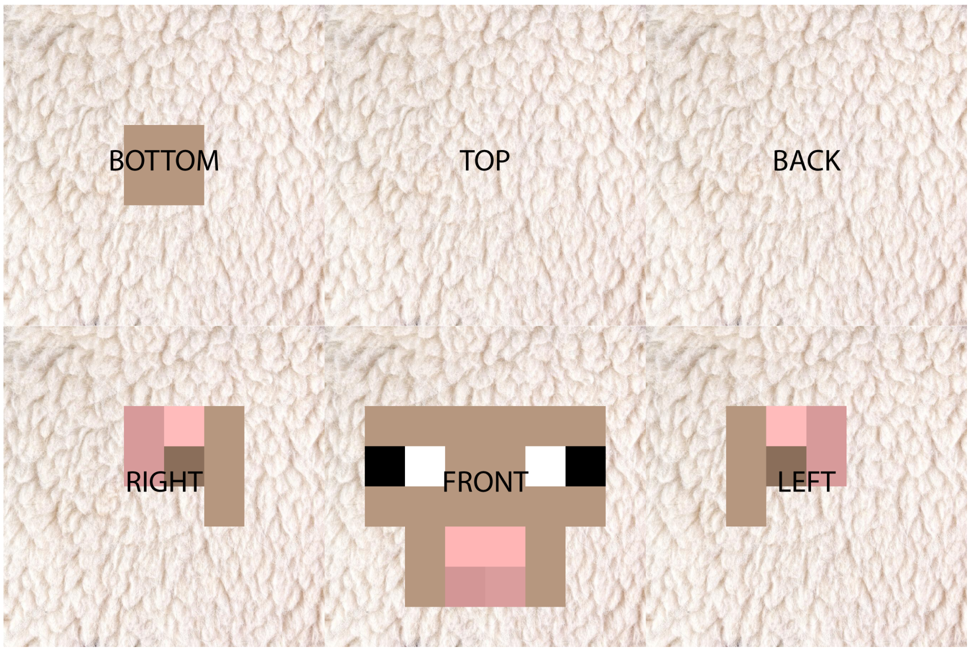
The Homework2 folder structure is organized as follows:

- **tools/**: In this folder there are the classes to implement the camera, the lights, the shapes, the objects, the backgroud and the animations.
- **textures/**: All the images to load are stored here.
- **shapes/**: In this folder there are all the shapes generated by extending Shape.js in the folder tools.
- **Homework2.js**: Contains the main class of this project.
- **Homework2.html**: The page of this project which contains also the fragment and the vertex shaders.

To generate the sheep and fence models, it was extended the Shape base class to create the CubeShape class. After creating the various shapes using the aforementioned class, the objects that make up the models were instantiated using the ObjectMaterial class. Next, using the method ObjectMaterial.addChildren(), it was created a hierarchical model of a sheep, composed as follows:

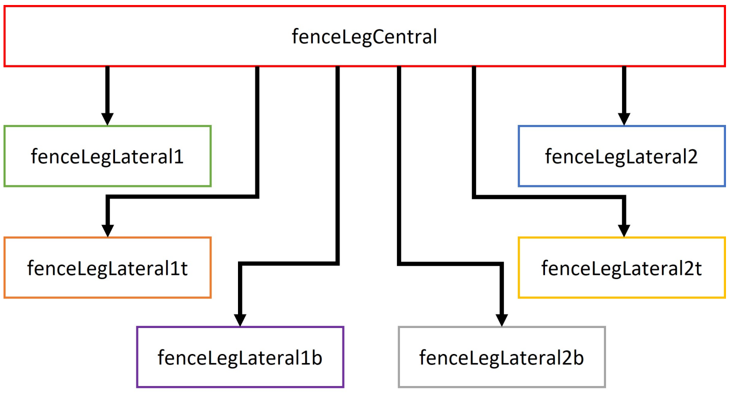


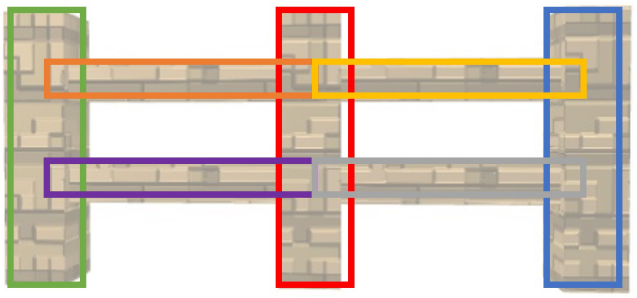
The sheep has 3 textures: legDown.png for the legBottom objects, sheepWool.png for the tail, the body and the legTop objects and headSheep.png for the sheep head. While the first two are computed in the same way as for the objects in the Homework1, the latter computes part of the image for each face, using a method similar to cube mapping. Below is explained how the texture image is divided:



The sheep and the fence were inspired by the models that are present in the videogame "Minecraft" (but with a more realistic wool).

The fence was also generated as an hierarchical model:

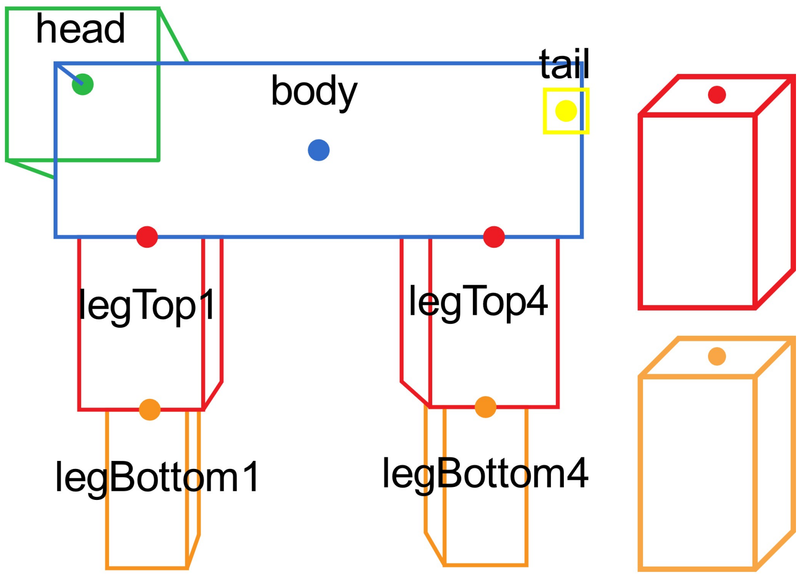




The main key of this project is how each object interacts with its children: in the render phase

(namely, ObjectMaterial.renderObject()) the modelMatrix generated by the parent class will be passed to its children: that way, every rotation and translation computed on the father will be propagated to its children.
Each object has 3 main properties to define its position in the space: theta, which is the angle of rotation in the x,y and z coordinate; position, which defines its translation in the 3D space and lastly centerOfRotation, which is nothing else than the pivot: it's the center of rotation of the object defined for the theta variable. With these 3 variables, each object is animated to compute the animation of the sheep.

Before computing the animation, all the starting positions of the two main objects (fence and sheep) are set and, more importantly, also their centerOfRotation. This is important for the sheep that will be animated; below it is visualized the position of the center of rotation of each object of the sheep (as a circle):

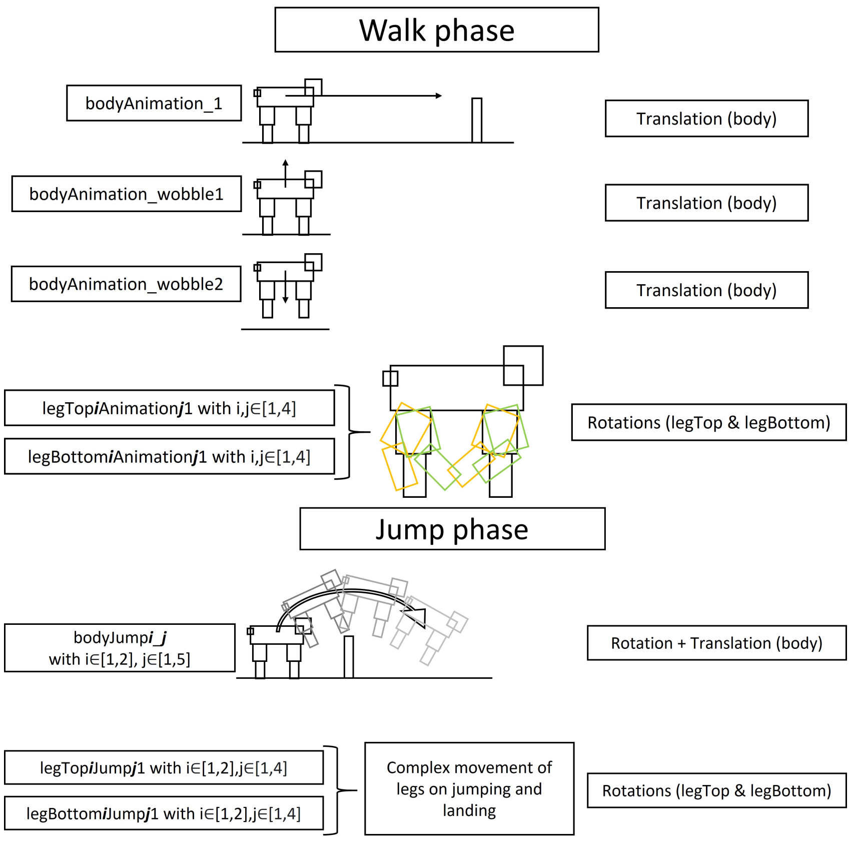


To understand how the animation works, firsly it is needed to know how AnimationObject and AnimationsManager interact with the ObjectMaterial class. The former defines a new translation and/or rotation in the 3D space, giving a starting position and/or rotation, a final position and/or rotation, the object that will do the animation and the necessary frames to compute the animation from start to finish. The class AnimationsManager is composed of multiple AnimationObject to create complex animations starting from simple ones and adding them to each other; it receives as arguments the animations, which is a 2D array: each row of the matrix corresponds to one or more animations to compute at the same time. Below is reported an example:

```
animations = [
    [anim11, anim12, anim13],
    [anim21, anim22, anim23],
    ...
]
is equal to: anim11+anim12+anim13;
when finished start anim21+anim22+anim23 ...
```

The AnimationsManager accepts also the endCondition parameter, that defines when (or if) the animation will stop. It also receives the onEnd() callback executed when the animation is finished and the delay to start the animation (number of frames to wait).
Finally, the ObjectMaterial class can receive as many AnimationsManager instances as it's needed (each of them will compute at the same time or when specified, to create even more complex animations if needed) using the method addAnimation(). In the renderObject method each AnimationObject (that will be called by each AnimationsManager) will generate a cumulative movement and/or rotation that will be added in the animationFrame variable present in the ObjectMaterial. Then the resulting animation for that particular frame will be executed and finally the animationFrame parameter will be resetted using the method flushAnimationFrame().

The animation can be decomposed and analyzed in two parts: the running part towards the fence and the jumping and landing part. The former is composed of two AnimationsManager for the body that gives the translation effect and a "wobble" effect on the body and one AnimationsManager for each legTop (each of them will compute the animation for the top and for the bottom part). Using the "positionEnd" condition on the body of the sheep, when the fence is reached, the callback function will be executed and it will start the second phase: the jump and the landing. From here, it's as in the first phase to create a complex animation using simple AnimationObject. Below there is a schematic summary of the various AnimationObjects:


Walk phase
bodyAnimation_1
Translation (body)
bodyAnimation_wobble1
Translation (body)
bodyAnimation_wobble2
Translation (body)
legTop*i*Animation*j*1 with i,j∈[1,4]
legBottom*i*Animation*j*1 with i,j∈[1,4]
Rotations (legTop & legBottom)
Jump phase
bodyJump*i_j* with i∈[1,2], j∈[1,5]
Rotation + Translation (body)
legTop*i*Jump*j*1 with i∈[1,2],j∈[1,4]
legBottom*i*Jump*j*1 with i∈[1,2],j∈[1,4]
Complex movement of legs on jumping and landing
Rotations (legTop & legBottom)

The bump texture for the sheep and for the fence are automatically generated using the provided textures for the models; this is thanks to the function createImageGrayScale() in the Shape base class (already explained in the Homework1).
The grass field is the same used in the Homework1 project.

The lights in the scene (3 in the cylinder and one infinitely far away) are defined in the same way as in the Homework1 project.

The Camera class is the same as in the Homework1 project; it is simulated the view and the movement of a 3D videogame: w,a,s,d to respectively walk forward, left, back and right with respect to the point where the camera is looking at in the x-z plane, space and shift to respectively fly up or down (in the y-axis). These controls are not only via keyboard, but there are also buttons on the screen.

The sky background is generated using the Background class, which needs as input a dictionary of cube mapping faces (positive X, negative X, positive Y and so on). This is an example:

```
cubeMapImage = {
    pY: imageTopFace,
    nX: imageBackFace,
    pZ: imageLeftFace,
    ...
}
```