Homework 2

Interactive Graphics

Alessandro Giannetti 1592006

2019-05-09

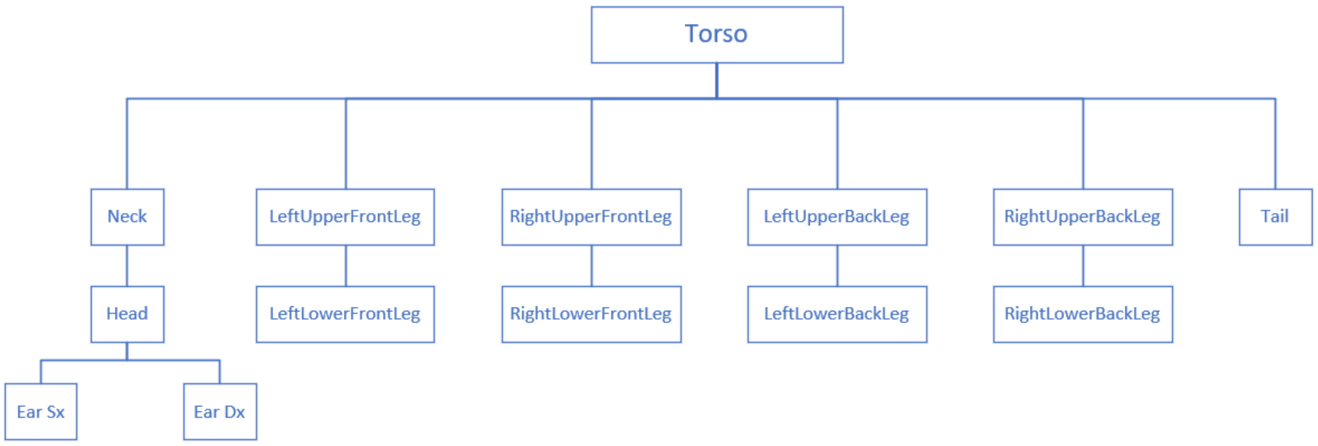
1 Introduction

This report describes the process of building a simple 3D interactive graphics project using HTML, JavaScript and WebGL. The application began with a simple humanoid hierarchical figure. In the next section, we will describe how the solution to various tasks is implemented.

2 Solution

Task 1: *Create a hierarchical model of a (simplified) horse, composed of the following parts; Body / Torso, 4 Legs (each one composed of 2 independent components ,upper and lower leg), Head and Tail. All components are cubes, use the cube function present in the file.*

A hierarchical model is a structure used to represent complex figures. In this model, there is one component which is the root, and the others are the children of the root and siblings of the children. To draw and render the whole figure, its needed to do a traversal visit and visit each node of the hierarchical tree. Each component, in this case, is a cube. So, for drawing a horse, the head of the horse is the root; the head, the tail, and the four upper legs are the children and the four lower legs are the children of the upper legs. I also decided to enrich the model of the horse by adding the neck and ears in order to make the model of the horse more realistic. Starting from the humanoid hierarchical figure, first of all I removed the sliders in the HTML file since theyre unnecessary (for the animation it will be used a specific function described later). Then, I changed the values of the theta vector in order to make this humanoid figure look like a horse. I also changed the dimensions of the width and height components. As mentioned earlier I wanted to add also the neck and ears, in addition to the tail. To draw them I added some variables: other slots in the theta vector for each element; and variables for storing the IDs, width and height; increased the *numNodes* variable. Its also needed to add a case to the switch of the *initNodes* function, which will set up the figure vector slot for the tail and call the tail() function, which is required for rendering the tail. Once this is all done, the *initNodes()* and *traverse()* functions inside the *render()* function will deal with the rendering and positioning of each component to create a horse figure.

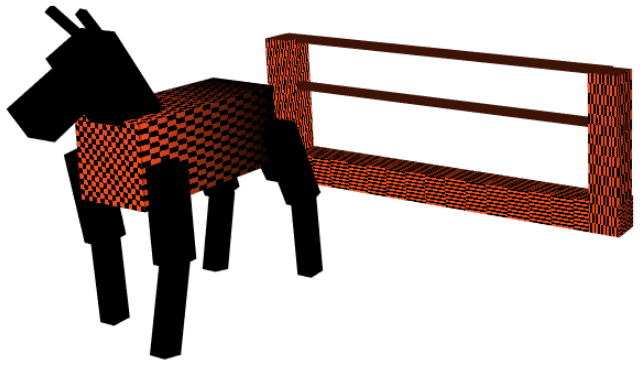


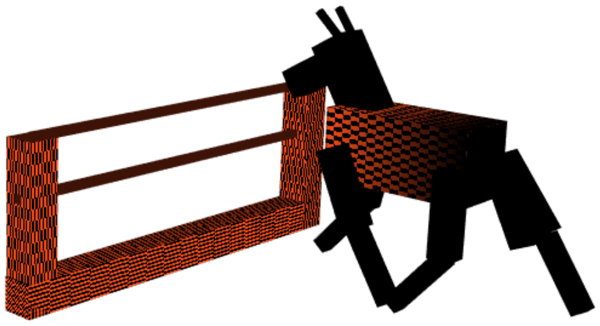
tree representation of a simple horse model

Task2: *Add a procedural texture to the body of the horse. The texture should be a checkerboard pattern but with a linear decrease of intensity from the front to the back of the body. Use, as a reference, textureCube4 of Chapter 7 of the examples of the textbook. Notice however that you should not apply a sinusoid but a linear decrease in the direction of the tail.*

For the second task, I create two array variables which will store the images for the textures. The *TcheckBoard* has a checkerboard pattern, like the textureCube4, while the *TcheckBoardDegr* is similar to the second texture of the textureCube4 but with a linear decrease (setting simply the I variable of the loop which goes from 0 to 255 linearly) instead of a sinusoid function. This will produce a linear gradient underneath the checkerboard texture.

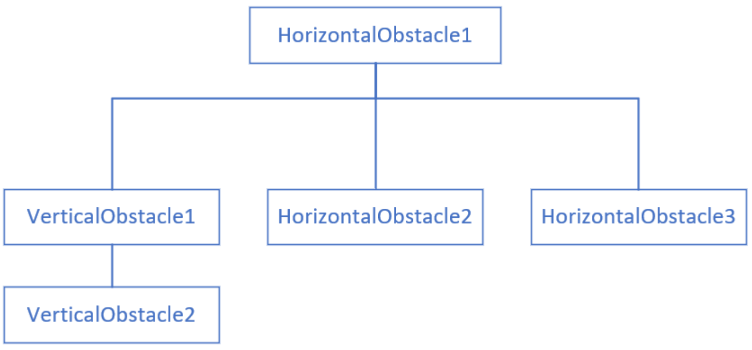
Then, I use the *configureTexture* function with a image parameter to generate respectively the two textures from the array images which have just been created, and pass them to the vertex and fragment shader via buffers (properly created and set in the *init()* function) in order to be applied to the body of the Horse. In order to keep the texture only in the torso, I called the configure texture function within the *torso()* function, after which through the parameter *fragTexType* I determined how the variable glFragColor should be calculated inside the vertex shader. In order to create the effect of colour degradation from head to tail by making sure to obtain a high-intensity chessboard on the front and a very dark one on the back In creating the torso cube I made sure to assign a specific texture to each face. If the face of the cube is the rear face, only the colour black will be assigned; in case the face of the cube is the front one, the colour orange + the texture will be assigned, taking care to pass as texture only the checkboard, while for the rest of the body as the front face the colour orange will be assigned with the texture composed by the checkboard and the degradation of colour.


Front view

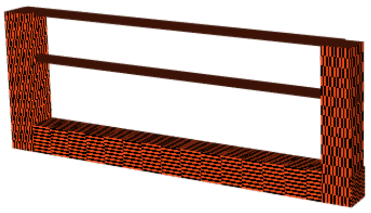

Back view

Task 3: *Create a (very simplified) model of a jump obstacle made with cubes. Use, as a reference, one of the following images (your version can be much simpler):*

To create a new object that represents an obstacle, I created a new hierarchical structure. The root of the model will be the first horizontal pole of the obstacle. tree representation of a simple obstacle model:



As for the horse, I added several variables, including the height and width of the segments that make up the obstacle and the various IDs that identify them. For each element has been added the value corresponding to the position of the ID within the theta array. having inserted this new hierarchical structure, I refactored the code. in particular, I separated the variables *numNodes* into *NumNodesHorse* and *numNodesObstacle*. then in the render function will be initialized a number of nodes equal to the sum of them. As for the addition of the tail, I added cases to the switch of the initNodes function for each element of the obstacle. For the obstacle I thought of applying a texture, but I didn't want to apply any color regression.



task 4: *Add a button that starts an animation of the horse so that, starting from an initial position where it is standing and positioned along the x axis, it walks by moving (alternatively back and forth) the legs, then jumps above the obstacle and lands after it.*

The first thing that I do is set the projection matrix in perspective so you can see better the jump of the horse. After that, I add a button in the HTML file that will start the horse animation. Then, its necessary to move the initialization of the nodes in the render function because each time a transformation has been done, the nodes need to be re-initialized and drawn again. I decided to divide the animation into several stages that are: *Walking*, *BeginJump*, *OnAir*, *EndJump* and *Reset*. In each function I have defined the animation for each part of the horse's body according to the action it must perform, established according to the distance of the horse from the obstacle. So, I create a bunch of variables that will store the values for rotations and translation.

The horse will have to move from right to left, and as soon as it reaches the obstacle it will jump over it. Walking the horse moves its legs alternately. As soon as the horse reaches the obstacle it will start to translate and rotate its torso upwards at the same time. The upper parts of the front legs will be extended, while the lower parts will be folded back on themselves. For the hind legs, however, will be extended backwards. the neck and head, on the other hand, will be slightly extended (by carrying out a rotation), as will the tail. As soon as the horse reaches the top of the jump, it will begin to descend gradually, simultaneously rotating and moving the body downwards. Before touching the ground will first extend the front legs, touching it, and then the back legs. During the ascent, the neck, head and tail will gradually return to their initial position. I also decided to make the horse come out of the size of the canvas, to make it start again (with the *reset* function), triggering a sort of "Pacman" effect, then bringing it back to the initial position.

Then, I use these variables inside the *initNodes* function to generate furthers transformation before the rendering. Last, I create a *animation()* function that changes these values each time the render function is called, making the necessary checks. Finally, I set the JavaScript function that starts the animation by clicking the HTML button. Whenever this button is pressed, if the horse its in its initial position it will start walking to the right, otherwise, it will reset all the values to default and the horse will be positioned in its initial status.

3 Conclusion

This solution has the main advantage of being modular and efficient. It correctly and efficiently implements different kinds of animation trying to emulate as faithfully as possible the jumping of a horse, as far as possible with a model horse composed of cubes. To observe better the animation on a different angle (as suggested on Piazza), I give the possibility, thanks to the sliders, to rotate the camera around the horse and the obstacle.