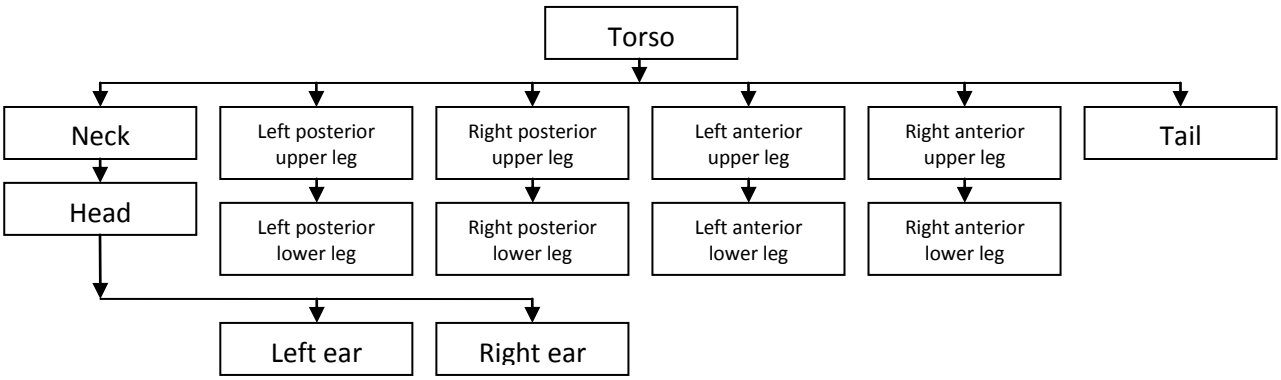
HOMEWORK 2 DOCUMENTATION

The goal of the second homework is to create a hierarchial model of a horse, and then to animate it so that it looks convincing and fluid enough while it runs and jumps across the obstacles on its path.

1. Creating the horse

Since the horse we want to create is a complex object, composed of various parts, such as the torso, the legs, the head and so on, that need to work alongside each other, we can use a hierarchial model in order to be able to depict it in a relatively intuitive way.
This model is composed by a series of interconnected nodes: each one represents a different part of the object, and contains a link to its right sibling (if there's any), a link to its left son (again, only if there's any), the name of the function that renders it on the canvas, and its own transform matrix, which details its position and orientation.
In our case, the horse is composed by the following pieces:

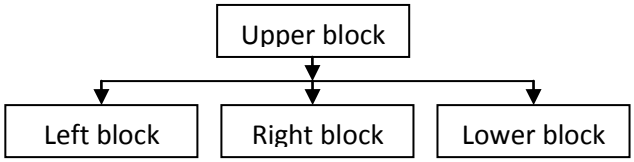


After an initialization process, we traverse through these nodes in a specific order by starting with the father of the tree, in our case the node that depicts the torso: in order to render the root, we create a new modelView matrix that is obtained by multiplying its transform matrix with the original modelView matrix (that we set at the start of the program), while pushing and preserving the old one in a stack so that it can be recovered at the end of the rendering process.
Then we continue to traverse the rest of the model with a recursive method, which accesses first the left child of the current node we are visiting, and later its right sibling. Since each node's position and orientation must be relative to the ones of its father, for the rendering of a child we need to set a specific modelView matrix that is obtained by multiplying the transform matrix of that node with the previous modelView matrix (which was previously used to render its father), while, again, saving the old one in the stack, so that it can be later popped and used by its siblings.

We do this for all the nodes until we reach the end of the model.
So, for example, when we need to render the left ear, we'll use the following modelViewMatrix:
leftEarMVM = originalMVM * torsoTM * neckTM * headTM * leftEarTM.
While, for the **l**eft **p**osterior **l**ower **l**eg, we'll use:
lpllMVM = originalMVM * torsoTM * lpulTM * lpllTM.
It must be noted that each part of the horse is represented by a cube, with its own set of values for its height, length and width, so that its proportions can be approximately similar to the real thing.

2. Creating the obstacle

In order to create the obstacle that the horse needs to jump over during its run, we'll use another hierarchial model, although simpler than the previous one:



Even though this model is separate (it even has its own traversing function), it's still linked to the previous one because the upper block has its position relative to the torso of the horse: that means that when we rotate the horse thanks to the slider defined in the HMTL file, the obstacle will rotate alongside it and will always be ahead of the animal, or behind it if the horse has already made the jump.
The reason the upper block isn't just another son of the "torso" node, but the root of a new model, is because we don't want to apply the torso's animations on the obstacle itself: otherwise the obstacle would oscillate while the horse runs, and, worse, it would jump alongside it when the animal makes the leap.
Instead, the transformation matrix of the torso is pushed into a second stack before any translation or rotation due to an animation is applied to it (but **after** it has been rotated with the slider), and then popped and multiplied with the transformation matrix of the upper block when it's time to render it.
It's imperative, so, that the horse is rendered before the obstacle, since we need the torso's translation matrix.

3. Animating the scene

When the button is pressed, the horse starts to animate and to run towards the obstacle: in order to achieve this effect, certain nodes, such as the torso and the legs, are refreshed with new rotation values before each frame is rendered. However these values must be inside a threshold, meaning they can't exceed limits on both the upper and lower ends: without

these precautions, the legs, for example, would perform 360° rotations, instead of moving back and forth, which is the effect we want.
One thing that must be noted is that the horse, while looking like it's running, remains still on its starting point: it's the obstacle to actually move towards the horse, thanks to a translation applied on the upper block that brings it closer to the animal frame by frame.
The jump animation begins when both of the objects are close enough, while the descent animation starts when the horse is exactly above the center of the obstacle: these animations are similar to the running one in the way they work, meaning that certain nodes (in this case the lower parts of the legs) are updated with new rotation values before each frame, with the difference that the torso keeps receiving new translation values to simulate the change of altitude.
This is an endless animation: once the obstacle moves out of the frame, it immediately "respawns" ahead of the horse, so that it can be jumped over again.
The animation can also be stopped by pressing the button again.

4. Texturing on the torso

The last task of the homework is to apply a linear vanishing effect to the checkerboard texture on the horse's torso, meaning that the texture loses its color intensity starting from the head and going towards the tail.
This vanishing effect is achieved thanks to a second texture that's multiplied with the checkerboard texture and the object's base color (brown) in the fragment shader: this texture has a starting color (rgb values: 15, 15, 15), which is linearly increased on each column with a small fixed quantity.
It must be noted that the front and the back of the horse are rendered slightly different than the rest of the torso, since the checkerboard texture is applied without the loss of intensity on the front, while it's almost and uniformly pitch black on the back: in order to this, we trigger the appropriate texturing process on the fragment shader with a global variable before we render each face of the torso.