Bear Animation
Homework for the Interactive Graphics Course
Università degli Studi di Roma La Sapienza

Matteo Russo 1664715

June 12th 2020

1 Introduction

This is the documentation for the second homework issued by Professor Schaerf for the Interactive Graphics Course.
This work introduces the animation of a Bear, built as a hierarchical model, that scratches its back against the cortex of a Tree.

2 Hierarchical Models

The Bear and the Tree were created as two separate hierarchical models. A hierarchical model can be seen as a "tree": we first define the root node and then we populate the other nodes as the children of the root one. It is also possible to define children of the root's children to expand the tree's width.
For the Bear structure we defined the "torso" as the root node and then the other parts as its children. Here's the structure:

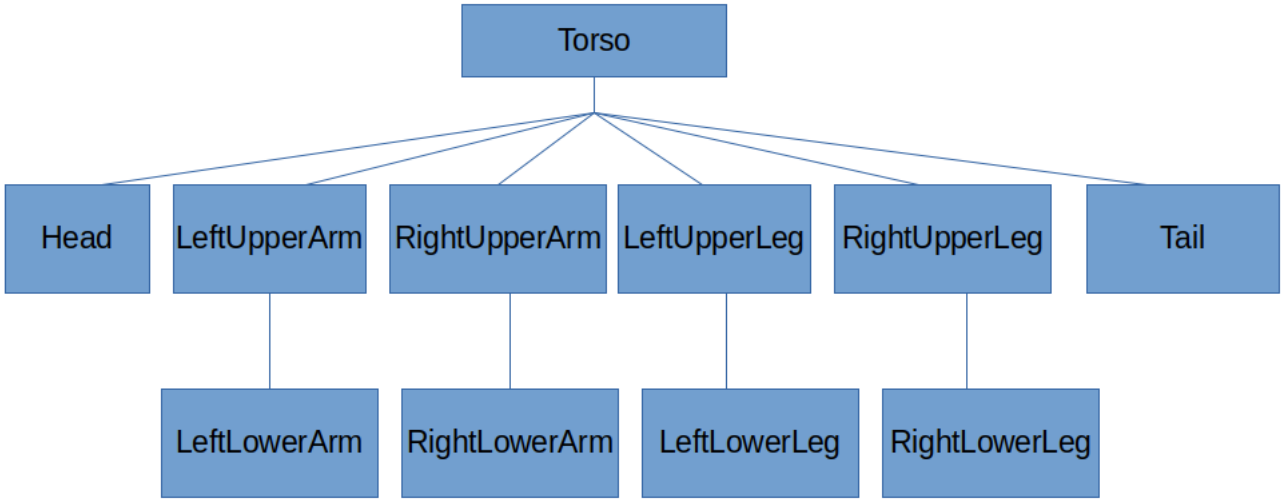

Figure 1: Tree Structure of the Bear

Definying a hierarchical structure for the bear was very useful when working on animations: by simply moving or rotating the torso node it is possible to apply the changes to all the other nodes. It will then permit to move the bear all the once.
The Tree has a more simple structure, just a root node formed by the trunk and a child representing the leaves.

3 Animation

All the rotations of the objects in the scenes were stored in an array theta. Theta has 16 entry, each one representing one node. As one may notice, we store more entry than the actual object in the scene; this is because some objects, like the head or the torso, need more than one rotation.
Since the code is similar for all the parts of the animation, we analyze just the starting animation.
To permit the animation, we change the entry in the theta array, then it is needed to upload the desired node by the function *initNodes()* passing as argument the part we want to update.
As we can see in Fig.2, we need to change the sign of the rotation variable everytime the angle done by the arms gets too wide. By changing it, we control also the legs' angle since we are using the same variable for the movement.
The Animation can be split in 3 parts:

- The Bear starts its walking as soon as the "Start Animation Button" is clicked. When it arrives near the tree starts to turn around giving the back to the trunk
- The Bear starts getting up on its feet and then begins the Scratching Animation
- The Bear stops scratching, returns on its feet and then walks away

```
  //Walking Parameters
  theta[leftUpperArmId]-=rotation;
  theta[leftLowerArmId]+=rotation;
  theta[rightUpperArmId]+=rotation;
  theta[leftUpperLegId]-=rotation;
  theta[rightUpperLegId]+=rotation;
  theta[tailId]+= tail_movement;
  if (theta[leftUpperArmId] < 40 || theta[rightUpperArmId] < 35) {
    rotation = -rotation;
    tail_movement=-tail_movement;
  }

//Refresh each node
initNodes(leftUpperArmId);
initNodes(rightUpperArmId);
initNodes(rightUpperLegId);
initNodes(leftUpperLegId);
```

Figure 2: Walking Animation Code

The Animation is dealt in two different functions called startAnimation and Scratching. This two function are called in the render as soon as the *start* or the *startScratching* boolean gets true. The boolean *start* is true when the button is clicked, while the *startScratching* boolean gets true as soon as the first animation ends. In the Scratching function is also treated the animation of the bear walking away through an *over* boolean.
The time the bear spends scratching is defined by the *time* variable, set to 8.5 by default. As soon as the bear's back touches the trunk, we start counting by instantiating a Date.time() object. The passage of time is controlled by the difference between this variable when the bear gets up and the *end_moment* variable. When the difference between *end_moment* and *start_counting* is greater than time, it means that *time* seconds are passed. The *over* boolean then gets true and can start the walk away animation.
It is important to note that the animation can be stopped anytime by clicking on the button once again.

4 Texturing

Four textures were employed in the scene. The Bear has two different textures, one for the fur and a different one for the head. The Tree has a texture for the trunk and a different one for the leaves. To choose what texture

```
void main()
{
    //Check what texture coords use depending on the flag
    if (uIsBodyTexture_active)
      fColor = texture(ubodyTexmap, fTexCoord);
    else if (uIsHeadTexture_active)
      fColor = texture(uheadTexmap, fTexCoord);
    else if (uIsWoodTexture_active)
      fColor = texture(uwoodTexmap, fTexCoord);
      else
        fColor = texture(uleavesTexmap, fTexCoord);
}
```

Figure 3: Fragment Shader's main

apply to each object, everytime the render plots a new cube, some boolean are passed to the fragment shader. By default, the boolean for the fur texture is always set to *true*((uisBodyTexture_active), while the ones related to the head, the trunk, and the leaves are set to false.
If we are in the case in which we want to draw the bear's head, it is only needed to set the fur boolean to *false*. While if we want to draw the texture for the trunk, we need to set the fur and the head texture to *false*. This can be seen in the handling of all the cases in the fragment shader as in Fig. 3.

Figure 4: Scene representing the Bear scratching the back

The above pic represents one frame of the final result.

5 References

- *https* : *//www.w3schools.com/howto/howto_js_countdown.asp*