Interactive Graphics - Homework2

Paolo Tarantino
1666228

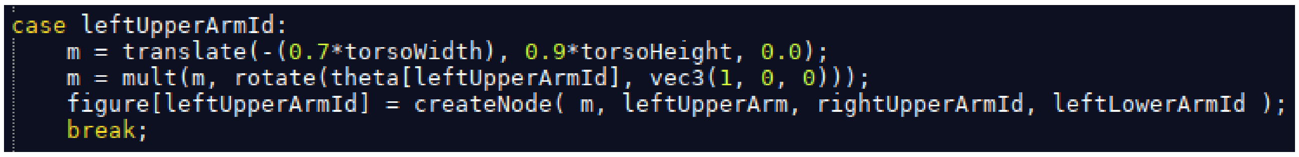
June 2020

1 Introduction

In this review I will give a report of the work done about the homework assigned. I will explain the target prefixed, which techniques are exploited and the effects that these have brought.
The required target was to create a hierarchical model of a Grizzly Bear that, through an animation, walks towards a tree and starts scratching its back against it (as shown in the figure below).

2 Models and Hierarchies

Starting with the human model provided by the professor, first I added a tail and changed the dimensions of the different parts, then I rotated and translated the figure to put it with the four legs on the ground, like a quadruped.
As the paper suggests, the bear is formed by a hierarchy having the torso as the root, and the head, the tail and the four limbs as its children. Every part is a cube (there is only a call at the *cube()* function, but the vertices initialized will be instantiated many times) that was scaled and placed in the correct position using the function *initNodes(Id)*, which creates a new node for each *Id* (overall 11 nodes for the bear), links up them to recreate a child-sibling structure, and associates a peculiar function to each component. For example the node of the upper part of the left (front) arm is created as below:

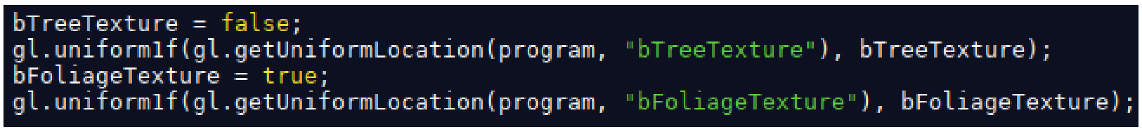


```
case leftUpperArmId:
    m = translate(-(0.7*torsoWidth), 0.9*torsoHeight, 0.0);
    m = mult(m, rotate(theta[leftUpperArmId], vec3(1, 0, 0)));
    figure[leftUpperArmId] = createNode( m, leftUpperArm, rightUpperArmId, leftLowerArmId );
    break;
```

where *m* is the matrix that will be applied for transforming the *ModelViewMatrix*, in this case it takes care of translating and rotating the limb opportunely; *leftUpperArm* is the characteristic function of this part, which draws the cube using the vertices declared at the loading of the page and shapes it also using the ModelViewMatrix; *rightUpperArmId* is the id of the sibling and *leftLowerArmId* the id of the child. All the nodes are stored in the *figure[]* array.
The tree is modeled as a separated object and is formed by a trunk and a foliage. The first one is obtained from an instance of a cube, the other one, composed by four triangles, has the shape of a pyramid (like a stylized pine tree). It is positioned on the right of the canvas at the same height as the bear.
These objects are re-drawn at each call of the *render()* through three other functions: *traverse(torsoId)*, which plots the nodes of the bear using the above mentioned structure; *tree()* which positions the trunk; *foliage()* which takes care of the leaves of the tree, and also models another surface to look like a ground, for making the scene more realistic.

3 Textures

Texture images are imported through html tags and set with the *configureTexture()* function in the javascript, and then sent back to the fragment-shader using the proper variables (e.g. uFoliageTextureMap). These are related with the *texcoords*, which are associated at each vertex in order to map the image on the object correctly.
There are two textures for the bear, one for the body and the other one for the head, and also the tree has a texture for the trunk and another for the leaves; the latter is also used for the ground.
Since there are different objects that need different textures, I created some boolean variables which will set to **true** or **false** according to the part that will be drawn. For example, before drawing the foliage I have:

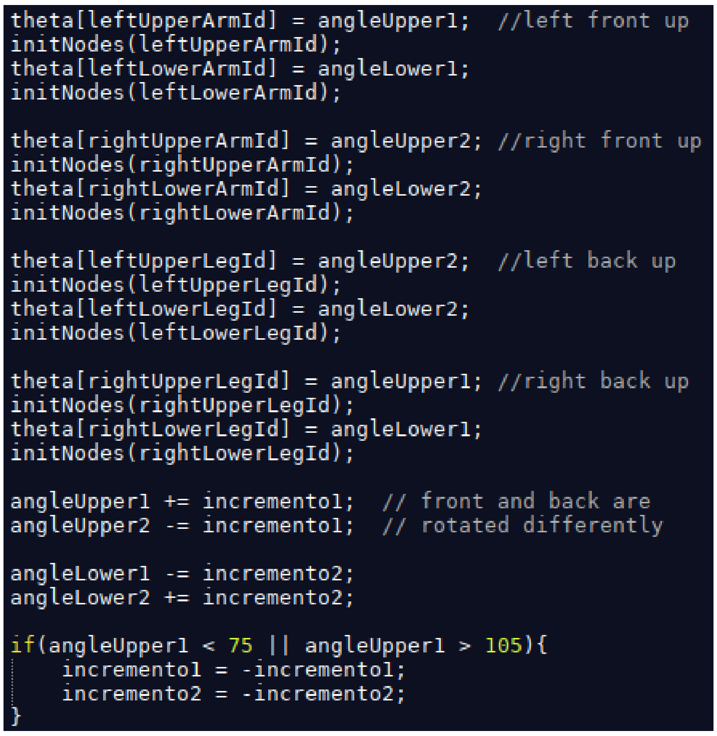


```
bTreeTexture = false;
gl.uniform1f(gl.getUniformLocation(program, "bTreeTexture"), bTreeTexture);
bFoliageTexture = true;
gl.uniform1f(gl.getUniformLocation(program, "bFoliageTexture"), bFoliageTexture);
```

where I notify the fragment-shader which is the correct texture to apply. In this shader I put some *if-else* conditions that read which value is set at **true** and associate the proper map to each fragment.

4 Animation

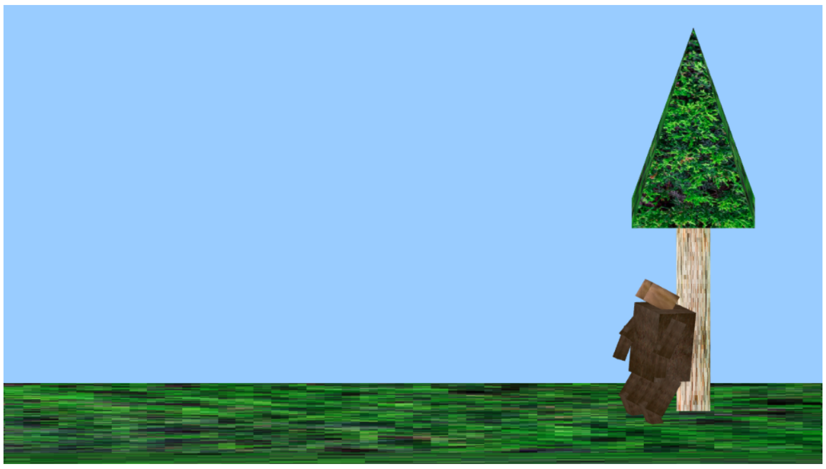
For the required animation, I decided to divide it in three different parts: walking, standing up, scratching.
With one click on the button *StartAnimation*, the bear in the left side of the canvas starts walking towards the tree. This animation is composed by several smaller movements: the torso is translated along the x axis increasing that component; the legs are rotated alternately back and forth (i.e. left front paw with the right back one, and the right front paw with the left back one), according to proper angles that are increased and decreased of specific quantities, and when the rotations reach the prefixed limit of the angles these quantities are turned at their negative values; also the head and the tail are rotated and moved to make the best realism. Thanks to the structure of nodes described previously, the children follow the translation and the rotation (movement that will be analyzed later) of their parent; so moving the torso, the other parts do the same.
In the picture below we can see the implementation of the movements of the four legs, which consists of continuous updates of the angle variables (thanks to the *render()* function) and the check of the width of angles.



```
theta[leftUpperArmId] = angleUpper1;  //left front up
initNodes(leftUpperArmId);
theta[leftLowerArmId] = angleLower1;
initNodes(leftLowerArmId);

theta[rightUpperArmId] = angleUpper2; //right front up
initNodes(rightUpperArmId);
theta[rightLowerArmId] = angleLower2;
initNodes(rightLowerArmId);

theta[leftUpperLegId] = angleUpper2;  //left back up
initNodes(leftUpperLegId);
theta[leftLowerLegId] = angleLower2;
initNodes(leftLowerLegId);

theta[rightUpperLegId] = angleUpper1; //right back up
initNodes(rightUpperLegId);
theta[rightLowerLegId] = angleLower1;
initNodes(rightLowerLegId);

angleUpper1 += incremento1;  // front and back are
angleUpper2 -= incremento1;  // rotated differently

angleLower1 -= incremento2;
angleLower2 += incremento2;

if(angleUpper1 < 75 || angleUpper1 > 105){
    incremento1 = -incremento1;
    incremento2 = -incremento2;
}
```

When the bear is near to the tree, it shifts a bit the direction to the right, in order to have the trunk at its side. So, it starts rotating the torso and standing up (around the y and z axes), while it is still approaching the tree. These movements are obtained increasing slowly the corresponding angles of the torso, and also angles of the legs to reach a similar position presented in the picture in the above introduction. All the angles of rotation of all the parts of the body are contained in the global array *theta[]*, that is used to update their width when it is necessary.
Once the bear is standing, the third animation has effect. It starts moving the torso up and down, with its back against

the trunk. Similarly to the legs during the walk, the torso is translated along the y axis of a certain quantity, and then in the opposite direction when some specific limits are reached. Also the back legs rotate to better accompany the movement of the body, and because they are formed by two distinction part(upper and lower leg), the are two types of rotation for each one. The front legs are raised forth and with the rotation of the head take part to the scratching movement, that also happens laterally, as a real bear does.



These three animations described above are set by the three functions *fWalking(), fStandingUp(), fScratching()* that are called in the *render()* function and are mutually exclusive. We can stop all the animation by pressing the button above the canvas.
The render function has the work to refresh the web page and call itself every time, so the variables and values are uploaded at each iteration based on the condition and the implementation in there.

5 References

- `https://en.wikipedia.org/wiki/Grizzly_bear`
- `https://youtu.be/thHZ9sWsXVg?t=146`
- `https://youtu.be/N3qTNuZCP8Q`