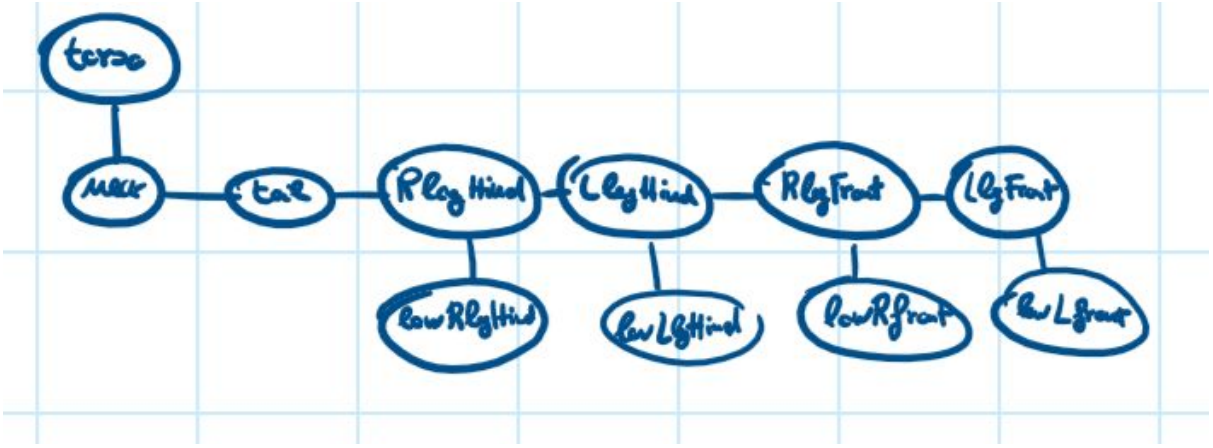
Leonardo Sarra - 02/05/2019

Homework 2 - Interactive graphics

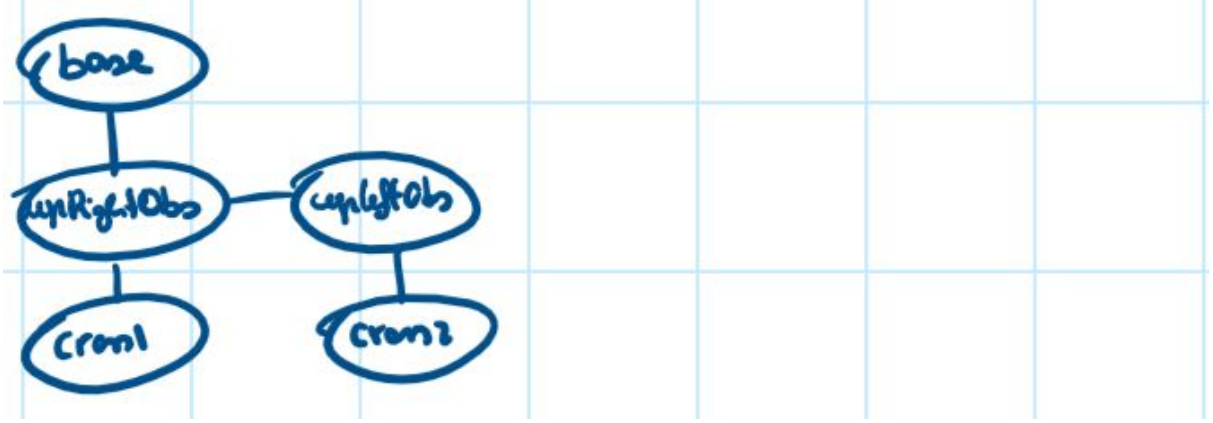
For this homework, I was asked to implement the model of a horse using WebGL.
This model was supposed to move on the scene with its leg animated and was supposed to jump an obstacle.
The horse's torso was also requested to have a checkerboard texture whose intensity decreased from the front to the back.
What follows is a deeper explanation of each of the feature that was requested.

Horse, obstacle and ground model

The specification required a horse model composed of different objects: legs (divided in the upper and lower leg), head, tail, and torso.
Those elements were put in an object hierarchy which helped me representing the complex model and to easily control it.
I decided to go a bit beyond what was requested and so I decided to add a neck to the horse in order to provide a more natural feeling to the representation.
In particular, the hierarchy for the horse's objects is as follow:



For the obstacle, I used the same approach but this time it was composed of: horizontal base, left vertical part, right vertical part, cross right to left, cross left to right and an upper horizontal part.
The hierarchy is organized like that:



Then at the end, there is the ground which is just a very flat and large green platform and so is composed of only one cube which got reshaped.

The hierarchy tree is then explored at run-time for rendering all the small objects that compose the scene, starting from root then siblings and eventually children.
Using the object hierarchy helped later on in animating the horse, if I didn't use that approach completing the task would have been a lot harder and with a high probability of breaking the model.
Note that the user can remove the obstacle from the scene by using an HTML button.

Animating the horse

The animation of the horse was done using a set of conditional checks which let me control the angulation on which the legs of the horse oscillates.
Each of the legs of the horse was animated separately in a similar way in order to provide a more natural result.
Moreover, some specific animations were developed for the horse's jumping and landing which will force the leg to take a horizontal position.
The body of the horse is animated to represent the push made with the legs at the start and then it is also animated during the landing for a more realistic effect.
Here is an example of animating a leg using conditional checks which are analyzed each time a render is executed.

```
    //left hind leg
    if (theta[leftLowerHindLegId] < 0) {
        leftLowerLegFlag = false;
    } else if (theta[leftLowerHindLegId] >= 60) {
        leftLowerLegFlag = true;
    }

    if ((isJumping && translationOverX > -6.3 && !isDescending) && theta[leftLowerHindLegId] <= 80){
        theta[leftLowerHindLegId] += 2.8;
    } else if (isDescending && translationOverX > 2 && theta[leftLowerHindLegId] > 80)
theta[leftLowerHindLegId] -= 0.7;

    if (leftLowerLegFlag && !((isJumping && translationOverX > -6.3 && !isDescending))) {
        theta[leftLowerHindLegId] -= 0.6;
    } else if (!(isJumping && !isDescending)) theta[leftLowerHindLegId] += 0.6;

    if (theta[leftUpperHindLegId] < 65) {
        leftUpperLegFlag = false;
    } else if (theta[leftUpperHindLegId] >= 95) {
        leftUpperLegFlag = true;
    }

    if (leftUpperLegFlag) theta[leftUpperHindLegId] -= 0.4;
    else theta[leftUpperHindLegId] += 0.4;
```

Note that the jumping is triggered at a specific distance along the x axis, when the horse is jumping a translation on both x and y axis is done, then, after a specific distance the horse will start the descending.
Animating using conditional checks is easy but you lose the ability to fine-tune the animation in a timely manner.
The resulting horse's animation works in a slow motion in order to make the movement more visible.
Another approach would be the one that consist of using keyframes with interpolation, that is supposed to provide better result but it's less immediate.
I decided to give the user the option to stop the animation and eventually to remove the obstacle so the horse will not jump and just walk on the scene, this was done in order to provide an environment in which the walk animation can be more easily seen.

Torso texture

The specification also required a 2D checkerboard texture on the body of the horse, moreover, the body of the horse was supposed to have its texture decrease in intensity in a linear way starting from the front to the back.
In order to do that the texture of the cube was split in multiple textures, each specific for a particular face of the cube.
The texture developed were:

- texture1: a clear texture for the front of the horse.
- texture2: a texture with a decrease intensity to use for the lateral part of the body.
- texture3: a black texture for the back.
- texture2_alt: an inverted texture2 for the upper part of the torso.

The content of the textures is declared as follow:

```
var image1 = new Uint8Array(4 * texSize * texSize);
var image2 = new Uint8Array(4 * texSize * texSize);
var image3 = new Uint8Array(4 * texSize * texSize);
for (var i = 0; i < texSize; i++) {
    for (var j = 0; j < texSize; j++) {
        var patchx = Math.floor(i / (texSize / numChecks));
        var patchy = Math.floor(j / (texSize / numChecks));
        if (patchx % 2 ^ patchy % 2) c = 255;
        else c = 0;
        // Clean texture
        image1[4 * i * texSize + 4 * j] = c;
        image1[4 * i * texSize + 4 * j + 1] = c;
        image1[4 * i * texSize + 4 * j + 2] = c;
        image1[4 * i * texSize + 4 * j + 3] = 255;
        // Black texture
        image3[4 * i * texSize + 4 * j] = 0;
        image3[4 * i * texSize + 4 * j + 1] = 0;
        image3[4 * i * texSize + 4 * j + 2] = 0;
        image3[4 * i * texSize + 4 * j + 3] = 255;
    }
}
for (var i = 0; i < texSize; i++) {
    for (var j = 0; j < texSize; j++) {
        var v = 0;
        var patchx = Math.floor(i / (texSize / numChecks));
        var patchy = Math.floor(j / (texSize / numChecks));
```

```
        if (patchx % 2 ^ patchy % 2) v = i;
        // Linear increase texture
        image2[4 * i * texSize + 4 * j] =  v;
        image2[4 * i * texSize + 4 * j + 1] =  v;
        image2[4 * i * texSize + 4 * j + 2] =  v;
        image2[4 * i * texSize + 4 * j + 3] = 255;
    } }
```

Then the content is pushed to texture variables that are later enabled for the shader to be used when the render traverse the torso element in the object hierarchy.
I opted for providing to the user an option to disable the texture from an HTML button.

Handling viewer position

In order to provide the best position for enjoying the animation of the horse, I had to introduce a moving camera.
For implementing this feature, I declared three variables: alpha, beta, and radius.
Those three variables were used to compute the variable “eye” which represents the camera position and rotates on the cartesian axis according to the variables’ values.

```
eye = vec3(radius*Math.sin(alpha), radius*Math.sin(beta), radius*Math.cos(alpha));
```

Moreover, the eye will be later used to compute an updated model-view matrix using the lookAt() function which transforms vertices from world space to camera space.

```
modelViewMatrix = lookAt(eye, at , up);
```

Instead of using the lookAt function we could do manually a series of normalizations and multiplication (replicating how lookAt works) but this approach is more complex and, considering that we are not interested in changing the lookAt’s behavior, it also doesn’t provide any advantage.
The user can control the viewer position dynamically from the HTML page, to do so a set of buttons were implemented.
Those buttons have attached a listener which changes at runtime the values of phi and theta in order to influence the eye variables in the render loop.