INTERACTIVE GRAPHICS HOMEWORK 2

A PREPRINT

Matteo Silvestri
Master in Engineering in Computer Science
Sapienza University Of Rome
Roma, Italia
silvestri.1774987@studenti.uniroma1.it

June 2, 2019

1 Introduction

This document describes the second homework assigned, the goal of this project is to create and animate a 3D geometric horse through the use of languages such as HTML, Javascript and WebGL. At follow i describe which kind of solution i have choose for implement this homework.

2 Explanation

In this project i use a orthographic projection, this kind of projection maintains parallel lines but provides no sense of depth, this kind of projection is used when we want an accurate representation of a model, to model this horse in the better way i use the Eye Camera, this kind of camera is defined by a position and a local coordinate system, we typically call the position of the camera the “eye” position(because it’s like the vision of a true human eye), the camera’s local coordinate system is defined by three orthogonal axes(x,y,z), with the "lookat()" function we can provide in the better way the "Eye Camera", this function required 3 elements:

- Eye location
- A center location
- An "up" vector

I use this function because the default projection in this frame is orthogonal, and in this way i can see in detailed manner the position of all the 3D objects and the movement of the horse, but this solution has disadvantages, like this:

- The “eye” and “center” point are the same location.
- The “up vector” has the same direction as the line-of-sight (which is a vector between the “eye” and the “center” point). For this case, the “same direction” includes both positive and negative directions. Stated mathematically, if the sine of the angle between the “up vector” and the line-of-sight vector is zero, which happens if the angle between the vectors is 0 or 180, the camera definition will fail.

For set the eye camera i used four different buttons two of them for Increase/Decrease the value of the "Theta" and another two for Increase/Decrease the "Phi", in this way a user can change in real time the view of the horse, and a developer can see the right positioning of all the objects. One of the goals of this homework was the implementation of a tail, for do this i manipulate in first time the "theta[]" vector adding a new parameter for changing the angle of the 3D Object, and in second time i created one more "case" inside the "InitNodes", in this function with a switch and an Id we can create and manipulate all the 3D objects, we can change the position of the tail for example with the "translate" function it takes three different parameters, and that represent the three axes(x,y,z), in all of the "case" we have another function for modelling the objects, that is the "rotate" function, it takes four parameters the first is the angle, the other three are the axes where we want the rotation to take place(their value can be 0 or 1, in the first case the rotation on that

```
case tailId:

m = translate(0.0, -0.3*torsoWidth, -0.25);
m = mult(m, rotate(theta[tailId], 1, 0, 0));
figure[tailId] = createNode( m, tail, neckId, null);
break;
```

Figure 1: Case Id of the Tail

```
function tail() {

    instanceMatrix = mult(modelViewMatrix, translate(0.0, 0.5 * tailHeight, 0.0 ));
    instanceMatrix = mult(instanceMatrix, scale4(tailWidth, tailHeight, tailWidth) );
    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(instanceMatrix));
    for(var i =0; i<6; i++) gl.drawArrays( gl.TRIANGLES, 0, numVertices );
}
```

Figure 2: Function of the Tail

axes is off, in the another one is on), beyond this we have for all components one external function where we can set the width and height of the object and where it is created. In the pic we can see an example. Another problem was the construction of the obstacle, because if we had used the same "IniteNodes" function and the same array for make the object(figure[]) when we animate one part of the horse the object of the obstacle of which is formed they would have the same animation, for solve this issue i created a new array for make the obstacle(obstacle[]) a new "InitNodesObs" function for handle all the request explained above and another function for the new object, i have created two column and two bar, and i add a new "traverse" function for initialize the "obstacle[]" array. For color the horse i use a texture showed below, this texture respect the condition of the homework, in the body the coloring of the torso tends to darken. Another request for the homework was the animation of the horse, for do this feature i manipulated the angle of the leg in this way, increasing the value of the angle by a value that is not too large, i was able to recreate the effect sought, the animation was triggered from a button inside the HTML file, when a user trigger the button the horse begin the movement and will go towards the obstacle, the animation management is all inside the "render" function. Now we can talk of the most bigger problem inside this homework, make horse jumping as fluid as possible, for solve this other issue for first i created a variable "torsoY", i initialized this variable with the default value of the position of the horse and of the obstacle in the Y axe, and i put this inside the translate function of the "torso", in this way i can solve the problem of the jump, with the management of the value of the angle i I made the jump as fluid as possible, with the variable "torsoLoc" i was able to understand the various points that the horse exceeded during its movement with respect to the X axis, in this way i was able to understand in which instant to make it jump and in which instant to make it land, trying to make the jump and the lending realistic as possible, when the horse comes in a certain position it restart the running and the value of the angle manipulated comes to the original value.

3 Conclusion

in conclusion we can see in the images below the moment of the jump, and through the use of the "eye" we can see at all angles the movement made by the horse and its relative jump:

```
var image1 = 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;
            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;
        }
    }

var image2 = new Uint8Array(4*texSize*texSize);


    for ( var i = 0; i < texSize; i++ ) {
        for ( var j = 0; j <texSize; j++ ) {
            image2[4*i*texSize+4*j] = i-255;
            image2[4*i*texSize+4*j+1] = i-255;
            image2[4*i*texSize+4*j+2] = i-255;
            image2[4*i*texSize+4*j+3] = 255;
           }
    }
```

Figure 3: Texture used

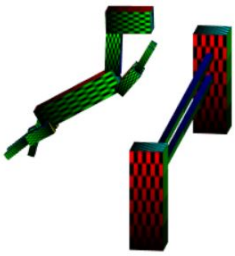

Figure 4: Jump

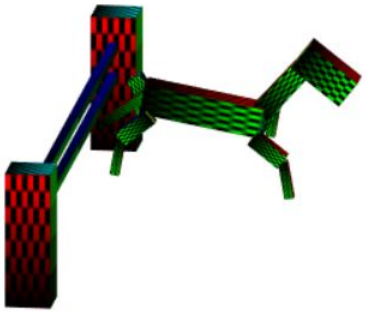

Figure 5: Landing