Brief description of the techniques, advantages, disadvantages and features

My approach to the problem followed a simple classic and mathematical way to solve various questions. The hierarchical structure of the horse was given almost completed and my work it's been the adding of the tail that it's built following given references. So it's been created a new node, a new drawing linked function and number of nodes had to be increased. Basically position and angle it's been fixed using a visual check.

To recreate a chessboard texture i had a look at previous homework basically editing number of checks, the way the texure is built modifiyng the quad(a,b,c,d) and cube() function and position of texture coordinates. In order to delete texture from the rest of the torso my approach was to use the deleteTexture() function.

Obstacles are recreated following a very simple recostruction of the original one, vertical and oblique rods are built using same techniques for the tail and position and angles are checked time by time. In this case a new hierachical tree is created where the base of the obstacle is the master of this tree. At the end of render function both torso node and onbstacle base is started if required.

```
traverse(torsoId);
if (show_obstacle)

    traverse(obstacle_Base_ID);
```

The animation is basically performed using a continuous increase of position of the torso that causes a movement of all structure according to its master role inside the hierarchical structure. Of course this is not the best solution to apply, in fact interpolation techniques are better in this case, but this simple approach worked and so i decided to go forward with it. As specified below animation is divided in three parts and three function for the pose of legs and arms were built.

Hierachical Model of an Horse

The hierarchical model of the presented horse is composed by the following nodes:

1. torso
    (a) head
    (b) left upper arm
        i. left lower arm
    (c) right upper arm
        i. right lower arm
    (d) right upper leg
        i. right lower leg
    (e) left upper leg
        i. left lower leg
    (f) tail

Each component specifies a drawing function and case-function to initialize the particular node, here it's figured all about tail node:

```
case idTail:

    m = translate(0, - 0.3 * torsoHeight, 1.5);
    m = mult(m, rotate(tailAngle, 1, 0, 0));
    figure[idTail] = createNode(m, tailf, null, null);
    break;


function initNodes(Id) {
    var m = mat4();
    switch (Id) {

    function tailf() {
        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.TRIANGLE_FAN, 4 * i, 4); }
```

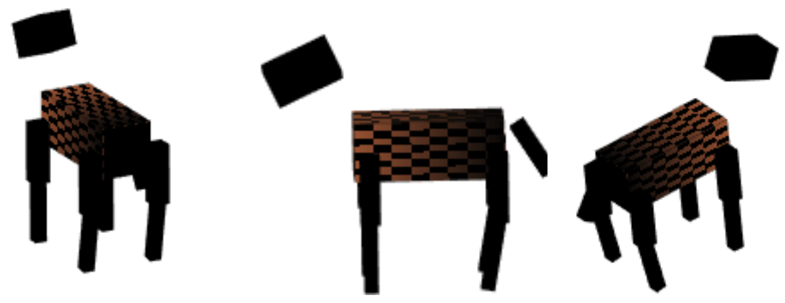
In fact for each component it's necessary to specify height, width, a numerical ID and an angle of rotation about one or more axis.

Chessboard texture applied just to the torso

The application of a procedural texture starts in the generation of a cheesboard image solution composed by two layers which will be combined thanks to the configureTexture() function:

```
var layer1 = new Uint8Array(4 * texDim * texDim);
...
var layer2 = new Uint8Array(4 * texDim * texDim);
    // scacchiera
    for (var i = 0; i < texDim; i++) {
        for (var j = 0; j < texDim; j++) {
        set = 255 - j; // decrease of intensity
        layer2[4 * i * texDim + 4 * j] = set;
        layer2[4 * i * texDim + 4 * j + 1] = set;
        layer2[4 * i * texDim + 4 * j + 2] = set;
        layer2[4 * i * texDim + 4 * j + 3] = 255; } }
```

The linear decrease of intensity is given using the variable set = 255 - j which shades to black. The result is shown in the figures below:



The texture is applied only on the torso using the deleteTexture() function inside the torso drawing procedure:

```
function torso() {
configureTexture(field)
instanceMatrix = mult(modelViewMatrix, translate(0.0, 0.5 * torsoHeight, 0.0));
instanceMatrix = mult(instanceMatrix, scale4(torsoWidth, torsoHeight, torsoWidth));
gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(instanceMatrix));
for (var i = 0; i < 6; i++) gl.drawArrays(gl.TRIANGLE_FAN, 4 * i, 4);
gl.deleteTexture(checkboard) }
```

In particular i got a black pattern on the back of the torso creating a new function (ass) which colourates black every pixel instead of brown: with this trick i got the result.

```
ass(3, 7, 4, 0); // sedere
function ass(a, b, c, d) {

    pointsArray.push(vertices[a]);
    colorsArray.push(vertexColors[1]); // black instaed of brown
    texCoordsArray.push(coordinateTexture[0]);
```

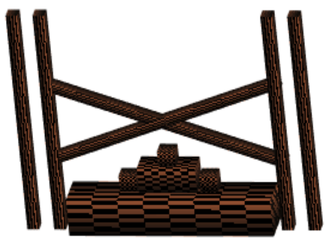
Obstacle

Choosen the third obstacles, the model is presented like a new hierarchical structure in which the horizontal base it's the master node of the tree, then we have 10 more nodes that complete the figure.

```
case obstacle_Base_ID:
m = translate(distance_from_target, -9, ground_quote);
m = mult(m, rotate(theta[(obstacle_Base_ID)], 0, 0, 1));
figure[obstacle_Base_ID] = createNode(m, obstacle_Base, null, obstacle_Circle_ID);
break;
```

1. var obstacle_Base_ID = 12;
2. var obstacle_rod_left2_ID = 20;
3. var obstacle_rod_right2_ID = 21;
4. ...

The result is shown below:



HORSE OBSTACLE

Animation and jump

When the user decides to start the animation the following loop is started and it's been divided in three parts depending on the position of the horse:

```
if (avviaAnimazione) {

    1) PART

    LegArmMovement();
    step += 0.35;
    camera_rotation = true;
    Zangle = 0.02; Xangle = 0.02; Yangle = 0.02;

    2) PART

    // start jump
    if (step >= 10 && step <= 45) {
        // equazione parabola della traiettoria
        step += 0.4;
        jumpHeight = (0.03265 * step * step) + (-1.7959 * step) + 14.694;
        jumpPosition();
    }

    3) PART

    // stop animation
    if (step >= 50) {
        finalPosition();
        avviaAnimazione = false;
    }
}
```

1. First part is when the horse is far from the obstacle, in this case legs are moved using LegArmMovement() function

   (a) The thought behind this function is to constantly increase the angle of the leg/arm until it's equal to a upper/lower bound, in that case the verse of rotation, which is expressed by run_rate is inverted. In fact at the beginning run_rate $= 1$, but when theta > 120, it's set to -1 and so on.

   ```
   // reach upperbound
   if (theta[leftUpperLegId] > 120)
   ```

```
        incremento_left = -run_rate;
    // reach lowerbound
    if (theta[leftUpperLegId] < 60)
        incremento_left = +run_rate;
        theta[leftUpperLegId] += incremento_left;
```

2. It's performed the jump when the horse is near the obstacle, legs are fixed using jumpPosition() function and height of the horse is calculated basically following equation of parabola:

   - $\mathrm{y} = \mathrm{ax}^2 + \mathrm{bx} + \mathrm{c}\ _{where\, x=step}$.

3. At the end of the jump, horse stops according to finalPosition() function.