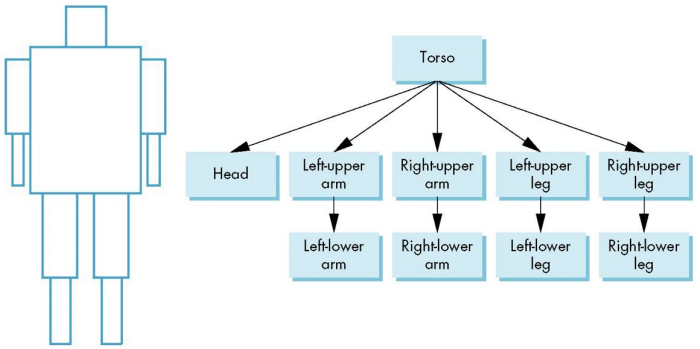
Progress Report
Homework 2

Lorenzi Flavio
mat.1662963
2018-2019

Il lavoro sostenuto per completare questo homework, è stato raggiunto grazie allo studio dei concetti studiati in classe e sul libro/slides, ma anche grazie alla continua e scrupolosa ricerca in internet dei concetti visti, in modo tale da approfondirli al meglio; per esempio è stato molto utile per il raggiungimento dei miei risultati il sito stackoverflow dove sono applicati concetti simili a quelli trattati per scopi diversi nel mondo di WebGl. Discuterò il mio lavoro punto dopo punto.

Punto 1: *hierarchical model of a horse (head, body, tail, 4 legs and 4 hooves)*

In questo punto ci è richiesto di implementare una struttura gerarchica già fornita (human body) e di trasformarla in quella di un cavallo, modificando i nodi già presenti (height, width, positions…) ed aggiungendone uno nuovo per formare la coda.



struttura simile a quella iniziale

-Nel file JS come prima cosa ho dovuto modificare i nodi relativi al torso aggiungendo in moltiplicazione a m (matrice di trasformazione) un'ulteriore rotazione tale da avere il corpo rivolto come quello di un cavallo.
Dopodichè ho effettuato modifiche simili ove richiesto negli altri nodi, avvicinando ogni gamba e zampa al busto (per esempio:
case gambaDxFrontId:
m = translate(0.333*bustoWidth +gambeFrontWidth, 0.9*bustoHeight, 0.0);), assegnandogli anche le dovute dimensioni (ingrossamento gambe e riduzione zampe per creare gli zoccoli).
In una struttura gerarchica ogni nodo è formato da **createNode(transform, render, sibling, child)**, dove in trasform va la matrice di trasformazione m, in render va la "creation function" da utilizzare per la parte del corpo che si vuole, e negli ultimi due nodi vanno rispettivamente fratello sx e figlio del nodo corrente: in tal modo è possibile creare oggetti "legati" uno con l'altro.
Quindi aggiungendo al set ID (giunti del corpo controllati da theta) il nodo coda e quindi aumentando il numero dei nodi presenti, ho inserito questo valore come sibiling di una gamba posteriore così da poter essere creato nel workspace: con la funzione **figure[codinaId] = createNode(m, codina, null, null);** sono riuscito a completare questo primo punto.
-Nel file HTML ho eliminato ogni slider presente e ho creato un bottone per la rotazione, collegato col file JS dove inserisco **modelViewMatrix = mult(modelViewMatrix , rotateY(30));**

Punto 2: *procedural texture with linear decrease of intensity*

-Nel file HTML ho eseguito tutte le operazioni necessarie alla creazione della texture **fTexCoord = vTexCoord;** nel vertex shader e **gl_FragColor = fColor*(texture2D(Top, fTexCoord)*texture2D(Down, fTexCoord));** nel fragment shader.
Ho seguito passo passo, come suggerito dal testo, gli esempi relativi alla texture per il cubo presenti nel codice eseguibile sul sito.
-Nel file JS, sempre seguendo il codice presente sul sito, ho eseguito i seguenti steps: 1)creazione del vertex color con aggiunta del colore marrone per il cavallo; 2)procedural texture simile a quella del sito, con scaling di intensità garantito per ogni facciata grazie a **b = 210 - j / 1.3;** anzichè solo 255; 3)creazione del vettore texCoord per garantire una certa forma alla texture; 4)implementazione della

funzione **quad** che per ogni vertice inserisce il marrone dal vertexColor e completa la creazione della texture compilando il vettore textCoordArray = [ ]; 5)funzione **fixTex** per la configurazione della texture (in cui implemento topT e downT ovvero le due texture da implementare) (richiamata poi dentro l'onload) e attivazione con **gl.activeTexture**. Per garantire la richiesta "texture implementata solo sul torso", nella funzione busto() ho quindi inserito **fixTex(topT)** e **gl.deleteTexture(downT)** per concentrare la texture solo in quella funzione e quindi solo sul torso. Infine per far si che tutto funzionasse ho dovuto inizializzare i nuovi buffers **cBuffer, tBuffer, vColor e vTexCoord** relativi a colore e texture.
Per concludere questo punto infine ho dovuto aggiungere le pareti trasparenti/mancanti con il comando **gl.enable(gl.DEPTH_TEST).**

Punto 3: *simple obstacle creation*

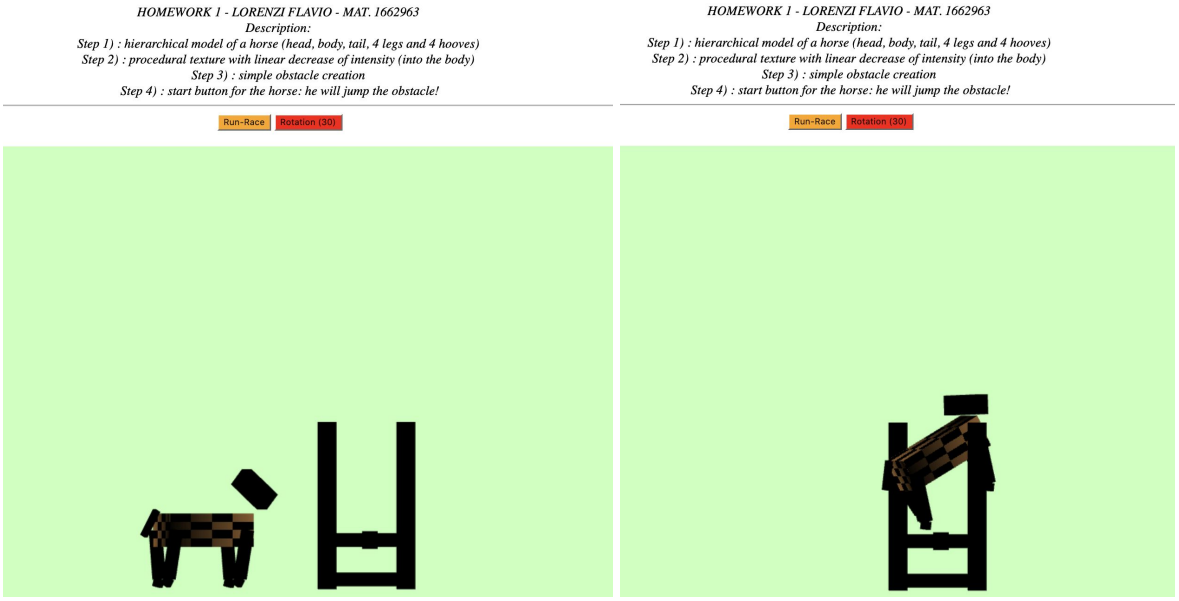
Per la creazione dell'ostacolo ho lavorato solo sul file JS:
il principio di creazione è quello utilizzato per la coda, in quanto ogni pezzo creato va ad implementare una struttura gerarchica dove abbiamo una base **obsMain()** che possiamo identificare come radice (come era il torso per il cavallo); da questa genereremo un figlio **figure[obsMainId] = createNode(m, obsMain, null, obsSxId);** che a sua volta avrà un fratello **figure[obsSxId] = createNode(m, obsSx, obsDxId, null);** e così via fino all'ultimo che invece avrà null nel campo sibiling.
Quindi implementando il set ID, theta, forma, dimensione e posizione di ciascun pezzo (le x,y,z cambiano a seconda della posizione del fratello "maggiore" da cui ogni nodo è determinato), ho creato un semplice ostacolo con due assi sx e dx laterali, una base centrale e due assi (superiore e inferiore) centrali.
Tra obsMain e busto del cavallo ovviamente c'è una relazione: per garantire una certa distanza tra i due oggetti (e per completare il punto successivo) ho posto una distanza **distanceFromOb** di mezzo, che in realtà specifica la posizione del cavallo rispetto all'ostacolo traslato di 12.
NB: sia per l'ostacolo che per il cavallo, nel render è implementata la funzione **traverse()** che permette la creazione di entrambi

Punto 4: *start button for the horse: he will jump the obstacle!*

Prima di tutto ho creato un bottone nel file HTML collegandolo alla rispettiva funzione nel file JS **document.getElementById("Go").onclick = function()** ; definendo una variabile di flag == false, ogni volta che il bottone è schiacciato si verifica che **goFlag321 =! goFlag321;** ovvero il flag cambia, in modo tale da avviare/fermare il movimento del cavallo ogni volta che si vuole.
Per completare quanto richiesto mi è bastato interagire con gli angoli dei giunti (ID) descritti da theta e nella funzione **go_Forward_And_Jump()** ho quindi implementato quanto segue: se il flag diventa true, allora avvia in primo luogo il movimento delle gambe e della coda creando un intervallo di valori per gli angoli associati (ogni valore oscilla tra lower/upper bound grazie a un incremento che viene ogni volta moltiplicato per -1 così da passare da -tot +tot); NB: per permettere un vero movimento del corpo è incrementata la variabile **distanceFromOb** (inizialmente pari a 0 nel codice, ma in realtà sappiamo essere -12 poiché l'ostacolo è traslato).
Per eseguire la parte del salto ho diviso in tre intervalli il momento(**"salita", "discesa" e "ritorno in posizione"**): *se la distanza è nel primo intervallo*, allora incrementa gli angoli di busto e testa (che si muoveranno di conseguenza) e la y del torso tale da farlo alzare a sufficienza; *se è nel secondo intervallo*, allora diminuisci le stesse variabili; *se supera infine un certo valore*, significa che il salto è compiuto e allora busto e yTorso ritornano standard.
Infine la funzione di jump va richiamata nel rendering.

Conclusione: questo lavoro mi ha immerso ancora una volta nel mondo di WebGl, questa volta alla scoperta dettagliata delle strutture gerarchiche e delle relazioni che le caratterizzano; è stato interessante sperimentare tutto ciò in prima persona, trasformando una struttura fissa e astratta in qualcosa di dinamico e avvincente come il salto di un ostacolo di un cavallo, interamente comandato da me. Ovviamente il risultato ottenuto assomiglia solo ad una vera simulazione, poiché lavorando solo con la funzione cube è difficile dare una reale forma al cavallo e, utilizzando solo gli angoli per l'animazione, è impegnativo assegnargli un movimento del tutto naturale.

Risultato:



Screen del file index.html