Interactive Graphics
Assignment 2

A.Y. 2017-2018

Hossam Arafat
1803850

Introduction

This report describes the process of building a simple 3D interactive graphics project using HTML, JavaScript and WebGL. The application began with a simple Hierarchical model of a dog consisting of a "Torso", Right and Left "Arms" and "Legs" split into "Upper" and "Lower" parts. The orientation of each body part can be controlled using sliders. We describe below as solution that we implemented to add several features to it.

Solution

It was required to add a "Tail" to the dog's "Torso". This was implemented by simply adding a "Tail" node to the Dog's Hierarchical model. The "Tail" node was chosen to be a child of the "Torso"; sibling to the "Head", "Arms", and "Legs". It was given an ID of 11 and a render function that specified its initial position and orientation at the rear end of the dog's "Torso".

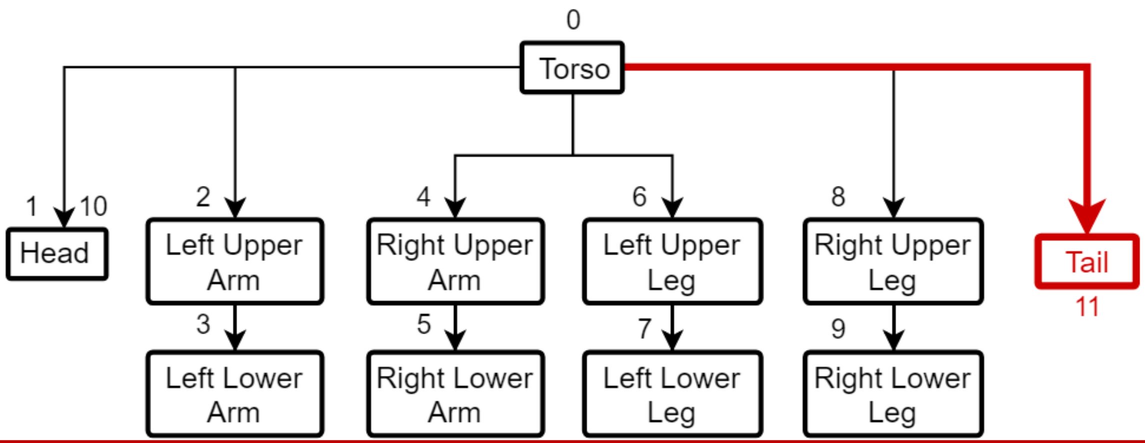

Fig 1: The updated Hierarchal Model of the Dog after adding a "Tail" Node (shown in red). The corresponding "ID" is shown at each node.

The second requirement was to apply a procedural texture in the form of a checkerboard pattern with a linear decrease in intensity from the "Head" to the 'Tail". This was achieved by generating 2 textures:

1. Checkerboard pattern texture
2. Simple Texture that varies linearly from bright to dark.

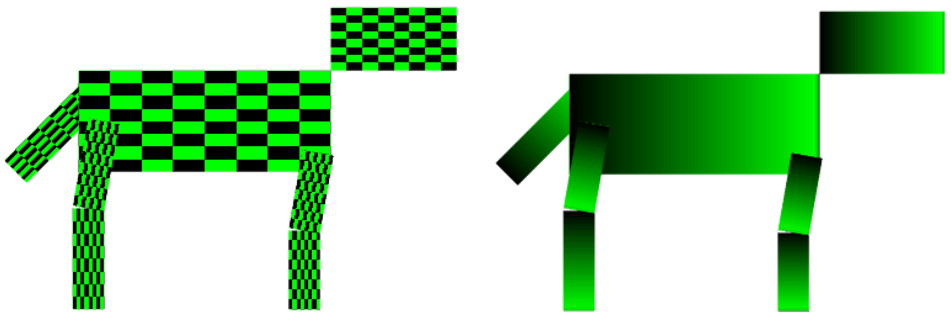

Fig 2: The checkerboard texture [Left image] and the linearly varying texture from bright to dark [Right image].

In general, the task of Texture mapping can be divided into 3 sub-tasks that take place in the JavaScript application, vertex shader, and the fragment shader:

1. A texture image is formed and placed in the texture memory of the GPU.
2. Texture coordinates are assigned to the vertices and consequently each fragment.
3. The texture is applied to each fragment.

Firstly, the 2D texture was formed by creating two 256 x 256 texture images in the application (which are essentially a 1D array of “texels”). The first image consists of an 8 × 8 checkerboard pattern of alternating white and black squares and the second image simply varies from “bright” white to “dark” gray. Rather than having to use integer “texel” locations that depend on the dimensions of the texture image, we parametrize the traversal of the texture image and use 2 floating point variables “s” and “t” that range from 0.0 to 1.0. In terms of the one-dimensional array of “texels” that we used in “gl.texImage2D”, the value *(*0.0, 0.0*)* corresponds to the Texel “Image1” [0], while *(*1.0, 1.0*)* corresponds to the Texel “image” [256*256-1].

We then create 2 texture objects, bind them and specify that the created images will be used as the current 2D textures. Regarding the texture parameters, after observing the effect of the different sampling and filtering techniques, ”Point sampling with Mipmapping” was applied to the checkerboard texture and “Nearest sampling with no filtering” was applied to the second texture as this combination –in our view– resulted in the least jagged checkerboard pattern with the sharpest-looking edges possible.

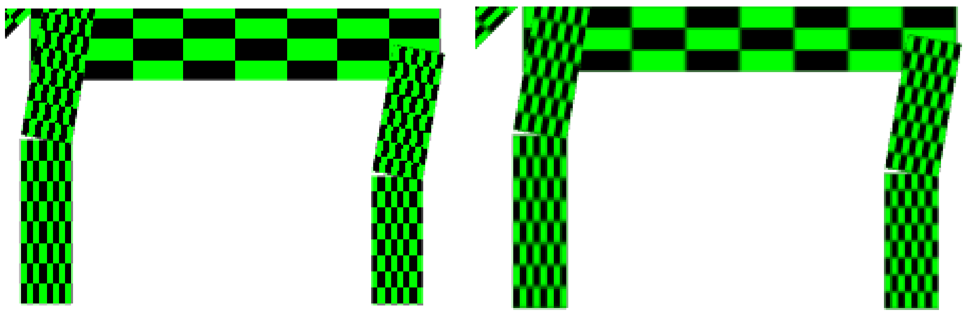

Fig 3: Nearest point sampling with no Filtering. [Left image] Point sampling with Mipmapping. [Right image]. The difference is especially clear in the “Arms” and “Legs”, where latter technique results in a less jagged albeit more blurry image. It is the author’s view that this result is overall better looking and more comfortable to the eyes.

Afterwards, the correct color for each fragment is acquired through a mapping between the location of each fragment and the corresponding location within the texture image. This mapping was achieved by assigning texture co-ordinates of our choice as an attribute to each vertex in the vertex shader, the same way vertex colors or normals are assigned as attributes to each vertex. Consequently, for each vertex in the (x, y, z) cube space, we assign (s, t) coordinates in texture space. In our application, we assigned the co-ordinates of the 4 corners of our texture image as attributes to the four corners of each face; namely: (0.0, 0.0), (0.0, 1.0), (1.0, 1.0), and (1.0, 0.0). We then simply push the Texture co-ordinates along with the vertex position and color inside our "quad" function.

Finally, we let the rasterizer interpolate the vertex texture coordinates to fragment texture coordinates. And within the fragment shader, we multiply the "color" attribute to the 2 "texture" attributes as a way of combining them to specify the final color of each face of the cube/"dog", and that is how we get an alternating "black & green" or "black & red" or "black & cyan"…etc. checkboard pattern with decreasing intensity from bright to dark across each face of our dog.

The final requirement stipulated that the dog should move across the screen, left to right while looking at the viewer. This was done by simply updating the "modelView" matrix in the application and initializing the nodes and incrementing the values for the "Arms", "Legs", and "Head" angles as well as the "Torso" x-position inside the render function.

Results

It is clear in Fig. 2 that there is a button to start and pause the animation and the dog successfully walks across the screen and looks at the viewer while doing so. The dog has a tail and a checkboard pattern texture of alternating "color" & black depending on the face of the cube. Moreover, it is most apparent in the torso that the texture decreases in intensity from the dog's head to its tail as is required.

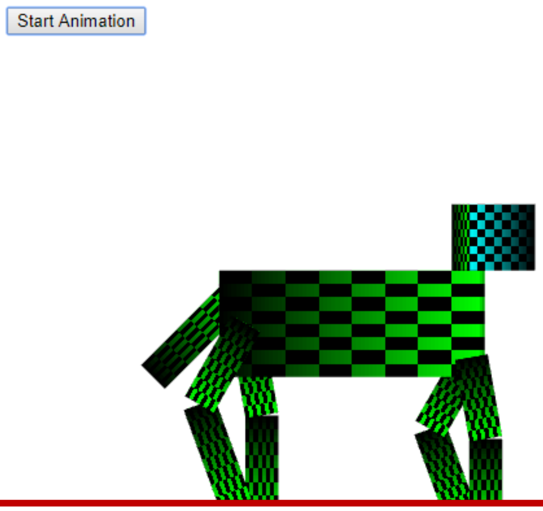

Fig 4: The final appearance of the dog. The dog has a tail, a checkboard pattern that is decreasing in intensity and successfully faces the viewer while walking across the screen from left to right.