CPSC314
Assignment 2: Fun with Transformations

Due 23:59:59, Oct 23, 2020

0 Plagiarism Policy

By handing this assignment in, you agree to the course plagiarism policy as stated here: `https://www.students.cs.ubc.ca/~cs-314/Vjan2013/cheat.html`. Please read it, and make sure you understand it.

1 Introduction

The main goal of this assignment is to practice rotation, translation, and scaling transformations. You will be responsible for creating, applying, and updating the transformation matrices in javascript without THREE.js matrix creation helper functions. However, the overall transformation matrices should remain relatively simple. There are two main tasks to be done: finish the octopus geometry and animate the arms.

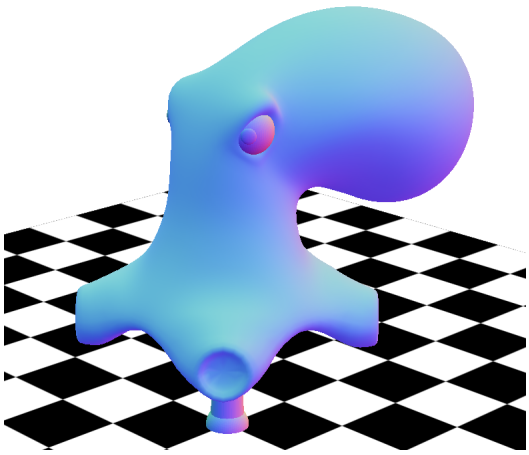

Figure 1: The provided template

The template code is very similar to the previous assignment template. The main differences are in the js file:

- The raccoon is no longer present. Instead, a part of an octopus has been constructed for you. There is the head (placed with respect to the head coordinate frame) and two pupils (placed with respect to each eye coordinate frame).
- A new incomplete function called `updateBody` is provided. It is called every frame, and you should complete it to animate the octopus.

2 Important Rules

THREE.js provides several tools for easy parenting, rotation, translation, and scaling of objects. Yet, as the purpose of this assignment is to understand the principle behind all of these methods, **you are not allowed to use them**. More specifically, you must:

1. Explicitly declare matrices using the `Matrix4().set` method. Creating matrices through operations from other existing matrices is also permitted (e.g., multiplication). Generating matrices through utility methods such as `Matrix4().makeRotationsX` is not allowed.
2. Objects must be moved by changing their coordinate frame using the `object.setMatrix` method. Operations on their `position, rotation, scale` attributes are not allowed.
3. Explicitly parenting objects using the `parent` attribute is not allowed.
4. Debugging: debugging in shaders and matrices is difficult, it is NOT TA's duty to debug your code.

You can find examples of matrix initialization and object positioning, as described above in the template code. In this assignment, as long as you respect the rules above, you can choose to implement from scratch as long as the final result looks similar.

3 Work to be done (100 pts)

First, ensure that you can run the template code in your browser. See the instructions from the previous assignment. **Remember to follow the requirements described in the previous section.** Study the template to get a sense of how it works.

1. **42 pts** Build arms
   Add four arms to the octopus. Each arm must be made of 3 joints and 3 links, and be placed with respect to the head coordinate frame. We have already created geometries of joints and links for you, your task is to implement `addOneArm()` function so that one arm is placed on the correct position. Firstly, let us add one arm on the octopus.
   - **7 pts** Add `joint1` on the octopus, it is a sphere rotated, transformed and placed as a proximal joint that connects the octopus socket and `link1`
   - **7 pts** Add `link1` on the octopus, it is a cylinder transformed and placed as a proximal link that connects `joint1` and `joint2`
   - **7 pts** Add `joint2` on the octopus, it is a sphere transformed and placed as a middle joint that connects `link1` and `link2`
   - **7 pts** Add `link2` on the octopus, it is a cylinder transformed and placed as a middle link that connects `joint2` and `joint3`
   - **7 pts** Add `joint3` on the octopus, it is a sphere transformed and placed as a middle joint that connects `link2` and `link3`
   - **7 pts** Add `link3` on the octopus, it is a cylinder transformed and placed as a middle link that connects `joint2` and it is the end of a arm.

   Note that we want the joints and links to be close enough to each other but do not overlap too much, so that the arm of the octopus does not disconnect and is not too short.

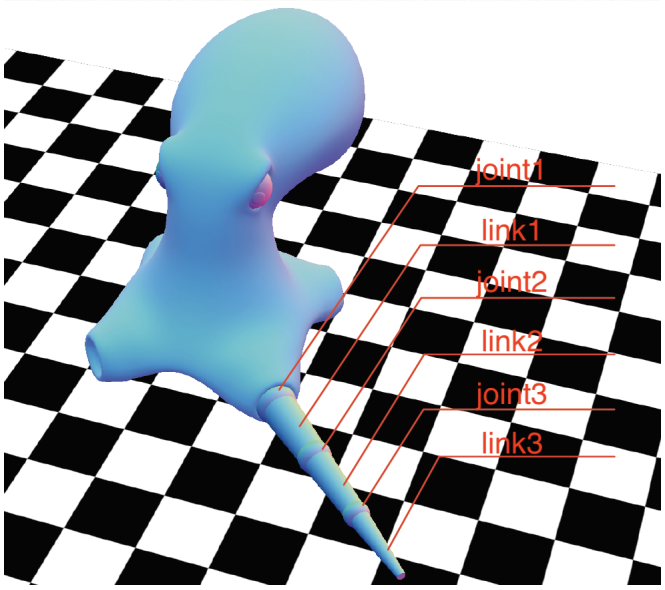

Figure 2: After adding one arm

2. **8 pts** Complete the octopus
   Now figure out how you can add four arms on the octopus. You can directly call your implemented `addOneArm` four times with your computed angles. Your result should be the same as below:

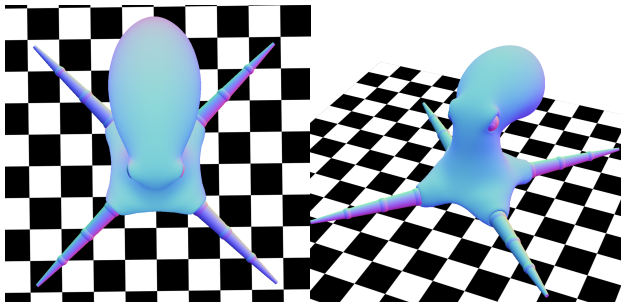

Figure 3,4: Octopus with four arms with side and upward views

3. **50 pts** Swimming octopus
   Now we want the octopus to move with time, but if you directly change `octopusMatrix.value` only the head of the octopus will move. You can press `3` on keyboard so that `updateBody` function will update frames and the octopus will move up and down periodically with time.

   (a) **10 pts** Animate eyes
   Firstly, we want the eyes to move with the head. As a example, we implement how can eyes rotate with the head, press `1` on the keyboard, and you will see that only head and eyes are rotating, and press `2` to stop. Your task to implement `updateBody` channel 3, and let head and eyes move together.

   (b) **30 pts** Animate one arm
   Copy and paste your code in `addOneArm` in `animateArm` and remove lines of `new THREE.mesh()` and `scene.add()`, you will find out that one arm will move with the octopus solidly. This is because the only difference when we update the body and build body is we do not need to create new meshes and add them to the scene since transformation matrices are the same.

Your task is to compute 3 rotation matrices with function of time, and rotate the links' coordinate frames so that they move like the octopus is swimming. Make sure the swimming animation is cyclical and continues to move back and forth. Also, make sure the octopus is moving at a reasonable speed to be easily observed.
The variable `t` *has been declared for you already and contains total elapsed program time. Obviously this value will always increase and we need a cyclical animation. You will, therefore, need a function to turn a constant value into a cyclical one and use it to change the orientation and position of the objects you want to animate.*

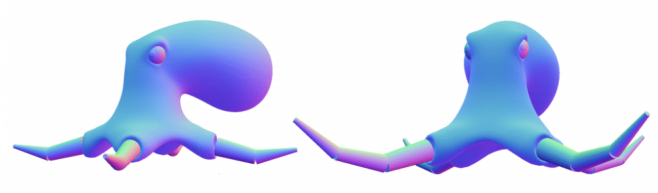

Figure 5,6: Example results for Q3

(c) **10 pts** Animate all arms
Now uncomment the function calls on `updateBody` so that we animate all four arms, again, you need to use the angles you computed in question 2.
You can view the example of final result in `Q3_result.mp4`

4 Bonus (4pts): Inverse Kinematics

Load `Octopus_08_A.obj` in js, and use inverse kinematics to make the octopus walk, your result should be similar to the result in `Octopus.mp4`, again your implementation should respect rules in section 2.

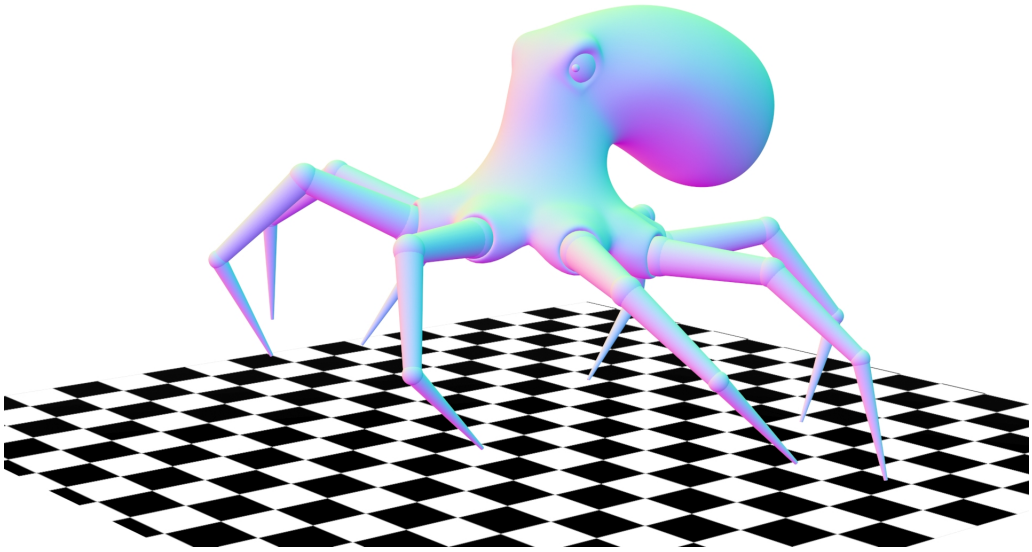

Figure 7: Walking Octopus (see the example video)