CPSC 314 Assignment 2: Transformations

Due on Feb 14, 2025

1 Introduction

In this Assignment you will utilize your knowledge of transformations to make things move. Here we will study how to build and animate with object hierarchies.

1.1 Getting the Code

Assignment code is hosted on the UBC Students GitHub. To retrieve it onto your local machine navigate to the folder on your machine where you intend to keep your assignment code, and run the following command from the terminal or command line:

```
git clone https://github.students.cs.ubc.ca/CPSC314-2024W-T2/a2-release
```

1.2 Template

- `A2.html` is the launcher of the assignment. Open it in your preferred browser to run the assignment, to get started.
- `A2.js` contains the JavaScript code used to set up the scene and the rendering environment. You will do most of your work here for this assignment.
- `glsl/` contains the vertex and fragment shaders for eye and sphere.
- `js/` contains the required JavaScript libraries. You do not need to change anything here.
- `glb/` contains the geometric model which needs to be loaded in the scene.
- `images/` contains the texture images used.

2 Work to be done (100 points)

Here we assume you already have a working development environment which allows you to run your code from a local server. (if you do not, check out instructions from Assignment

1 for details). Once you have set up the environment, Study the template to get a sense of how it works.

a. **(5 points)** Load the glTF armadillo model.

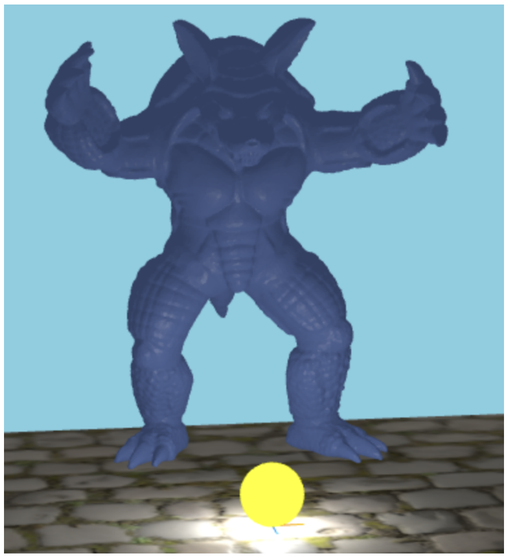

Figure 1: Question 1.a. Starter scene.

In this assignment, we'll shift from a OBJ model to a glTF model which is more powerful. A rigged glTF armadillo is provided in `glb/`. Your task is to load the armadillo into the scene as in Figure 1. Specifically, fill in function `loadAndPlaceGLB()` in `js/setup.js` and make necessary changes to `A2.js`. You can check THREE.js's website for glTF loader and function `loadAndPlaceOBJ()` for reference.
Resource: https://threejs.org/docs/examples/en/loaders/GLTFLoader

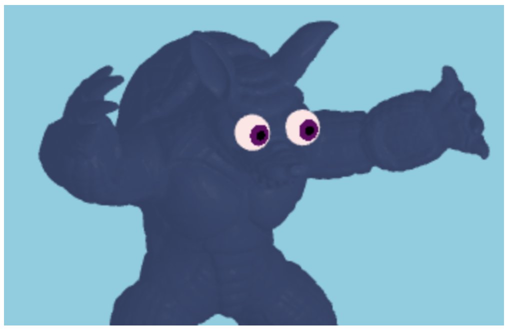

Figure 2: Question 1.b: Give the Armadillo eyes.

b. **(15 points)** Armadillos have Eyes. Your task here is to bless the Armadillo with a pair of googly eyes. One eyeball model is already provided. Your job is to place two copies of this eyeball onto where the eyeballs should be on the Armadillo using an appropriate `modelMatrix`. The result should look similar to Figure 2.

Hint 1: You can do this entire question in `A2.js`.
Hint 2: Don't worry if the eyes are initially pointing backwards into the Armadillo's head; in the next question you will have the opportunity to orient them correctly.

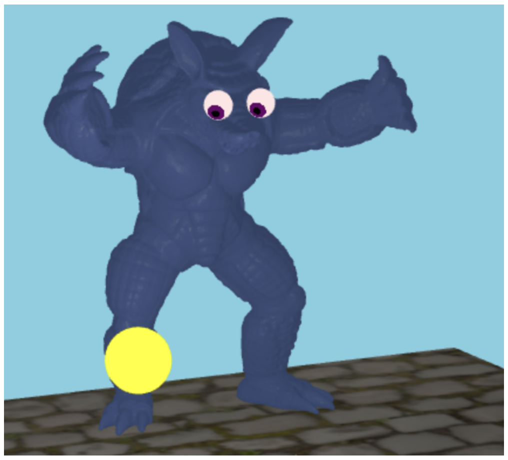
Figure 3: Question 1.c: Rotate the eyes to look at the sphere.

c. **(20 points)** Staring at the sphere. What good are eyes if they don't `lookAt` anything? Direct the Aramdillo's gaze towards the nefarious floating sphere. Ensure that the eyes look at the sphere as it moves around the scene as shown in Figure 3. Also, notice that in Figure 3 each eye contains a pupil and an iris that help you with verifying your `lookAt` works.

   Hint 1: `THREE.Matrix4 and THREE.Object3D` have a method called `lookAt` that may be of use.

d. **(20 points)** Laser eyes! Arm the Armadillo with eye lasers. Shoot the lasers from the Armadillo's eyes at the sphere when it gets too close. A distance threshold (units in world space) named `LaserDistance` is provided in `A2.js`. To be more specific, create two lasers from the eyes to the sphere. You need to create the geometries and shaders by yourself. Figure 4 shows a possible result.

   Hint 1: You may use any kind of material for the lasers (even a `ShaderMaterial`).
   Hint 2: Note that the distance threshold between the sphere and the eye is given in the world frame, but the scale of the laser will likely be in a different frame.

e. **(25 points)** Catch the Ball! Using *Inverse Kinematics*. Make the Armadillo move its left arm to try to catch the ball if it gets too close to the left arm. By "too close" we mean when the orb reaches inside a cone, have the armadillo reach for it. Figure 5 shows how this shold look.

   Hint 1: To animate the Armadillo, you may want to access bones within the Armadillo's skeletal structure. To do this, you can make changes at where you called `loadAndPlaceGLB()`.
   Hint 2: To define the cone which triggers the reaching action, use the bone defined by the base of the arm as the starting point and the direction of the arm as the axis.

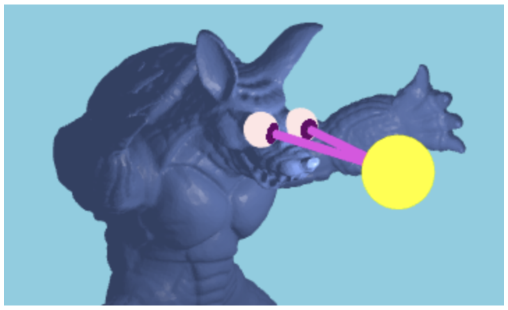

Figure 4: Question 1.d. Give the Armadillo laser eyes!

Hint 3: To visualise the bones responsible for inverse kinematics use the `IKHelper` module
Resource: `https://threejs.org/docs/examples/en/animations/CCDIKSolver`

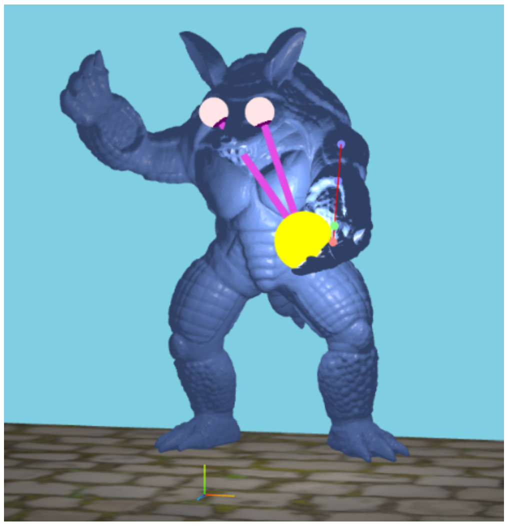

Figure 5: Question 1.e. Implement inverse kinematics!

f. **15 pts** Feature Extension.

In this part, you are required to extend the assignment to add a feature of your own choosing. The goal is to encourage you to explore the capabilities of Three.js and WebGL. For full credit for this part, the feature does not have to be complex or creative, but must be non-trivial (e.g., not just changing a color). Roughly requiring about 10 lines of new code. Some possible suggestions are:

- add interesting objects to the scene.
- animate the armadillo in other ways.
- animate colors, lights, in fun ways.
- play with textures, shapes, and movements.

For this question, first duplicate your current work into a new directory named 'part2', then implement your feature in this new directory. Write a brief description of your feature in the README file. You will be graded on both how it works and how well you can explain your feature.

3 Bonus Points

You have many opportunities to unleash your creativity in computer graphics! In particular, if you create a particularly novel and unique feature extension, you may be awarded bonus points. A small number of exceptional extensions may be shown in class, with the student's permission.

4 Submission Instructions

4.1 Directory Structure

Your submission should contain two subdirectories - the first should be named 'part1' and should contain all parts before the feature extension; the second subdirectory should be named 'part2' and should contain your feature extension. For each of the two subdirectories, include all the source files and everything else (e.g. assets) needed so each part can be run independently. Do not create more sub-directories than the ones already provided.
You must also write a clear `README.txt` file which includes your name, student number, and CWL username, instructions on how to use the program (keyboard actions, etc.) and any information you would like to pass on to the marker. Place the file under the root directory of your assigment.

4.2 Submission Methods

Please compress everything under the root directory of your assignment into `a2.zip` and submit it on Canvas. You can make multiple submissions, but we will grade only the last one.

5 Grading

5.1 Face-to-face (F2F) Grading

Submitting the assignment is not the end of the game; to get a grade for the assignment, you must meet face-to-face with a TA in an 8-min slot during or outside lab hours, on Zoom, to demonstrate that you understand how your program works. To schedule that meeting, we will provide you with an online sign-up sheet. Grading slots and instructions on how to sign up will be announced on Canvas and on Piazza. During the meeting, the TA will (1) ask you to run your code and inspect the correctness of the program; (2) ask you to explain parts of your code; (3) ask you some questions about the assignment, and you will need to answer them in a limited timeframe. The questions will mostly be based on ThreeJS or WebGL concepts that you must have come across while working on the assignment; but also you may get conceptual questions based on lecture materials that are relevant to the assignment, or technicalities that you may not have thought about unless you really "digged deep" into the assignment. But no need to be nerveous! We evaluate your response based mainly if not entirely on the coherence of your thoughts, rather than how complete or long your response is: e.g. if the full answer to a question includes A+B+C+D and you mentioned A+B only in the provided time, but your thread of thought is logical then you may still get full marks.

5.2 Point Allocation

All questions before Feature Extension has a total of 85 points. The points are warranted based on

- The functional correctness of your program, i.e. how visually close your results are to expected results;
- The algorithmic correctness of your program, e.g. applying transformation matrices in the right order;
- Your answers to TAs' questions during face-to-face (F2F) grading.

The Feature Extension component is valued at 15 points. You will earn some points as long as you implement at least one extension that the grading TA considers novel and unique. However, full marks may not be awarded if your implementation introduces significant visual artifacts that diminish the product's appeal, or includes critical bugs that render the system unusable (e.g., failing to respond to user commands). Exceptional feature extensions may also be eligible for bonus marks, allowing for a score exceeding 100 on an assignment. However, note that your overall course grade will be capped at 100.

5.3 Penalties

Aside from penalties from incorrect solution or plagiarism, we may apply the following penalties to each assignment:
Late penalty. You are entitled up to three grace (calendar) days in total throughout the term. No penalties would be applied for using them. However once you have used up the grace days, a deduction of 10 points would be applied to each extra late day. Note that

1. The three grace days are given for all assignments, **not per assignment**, so please use them wisely;
2. We check the time of your last submission to determine if you are late or not.

No-show penalty. Please sign up for a grading slot on the provided sign-up spreadsheet (link will be posted later on Piazza) before the submission deadline of the assignment, and show up to your slot on time. A 10-point deduction would be applied to each of the following circumstances:

1. Not signing up a grading slot before the sign-up period closes. Unless otherwise stated, the period closes at the same time as the submission deadline.
2. Not showing up at your grading slot.

If none of the provided slots work for you, or if you have already missed your slot, follow instructions outlined in this Piazza post to get graded after the F2F grading period ends. Also, please note that

1. you'll need to explain to your TA why you're getting graded late, and may be asked to present documents to justify your hardship. The TA may remove the no-show penalty as long as he/she deems the justification to be reasonable.
2. In the past some students reported that their names disappeared mysteriously due to technical glitches, and the spreadsheet's edit history has no trace of it. So double check that your name is on the sign-up sheet after you sign up by refreshing the page.