CPSC 314
Assignment 2: Armadzilla Attacks!
or Fun with Transformations

Due 11:59pm, Feb 10, 2014

1 Introduction

The main goals of this assignment are to explore vertex shaders a little more and practice some rotation, translation, and scaling transformations. You will be responsible for creating some new shaders this time around, though they should remain relatively simple as in assignment 1. There are two types of tasks to be done: first, add sphere-shaped eyes to the armadillo that track and shoot lasers at the gem; second, perform a simple rotation of the armadillo's head, to make him nod in satisfaction.
Since the midterm exam is on Feb 7, we have moved the deadline by a few days to give you some flexibility. But we *strongly* encourage you to finish it before the midterm. Doing the assignment is one of the best ways to understand transformations.

Template: The template code is found in the main assignment directory. It is very similar to the previous template. The main differences are:

- new variable `g_eye` that holds the mesh data for an eye
- new variable `head_rot_X` that holds the rotation angle for the head
- new variable `Leye_pos` that holds the left eye position
- new variable `Reye_pos` that holds the right eye position
- new variable `neck_pos` that holds the position of the pivot about which the head rotates

There is also a new `mode` variable, that lets you work on the three specified parts of the assignment below. You can cycle through the three modes (`MODE_EYES`, `MODE_LASERS`, `MODE_HEAD_DEFORM`) with the 'm' key.
You will need to edit the shaders and create some new ones for this assignment. When adding a new shader, read how `main.cpp` loads and initializes its other shaders and follow suit.

Important Caveat: when you first run the template you may not see anything because the shaders for the first two modes have not yet been written. Don't panic. Just hit 'm' twice to get to `MODE_HEAD_DEFORM`; you will see your familiar Armadillo.

2 Work to be done (100 pts)

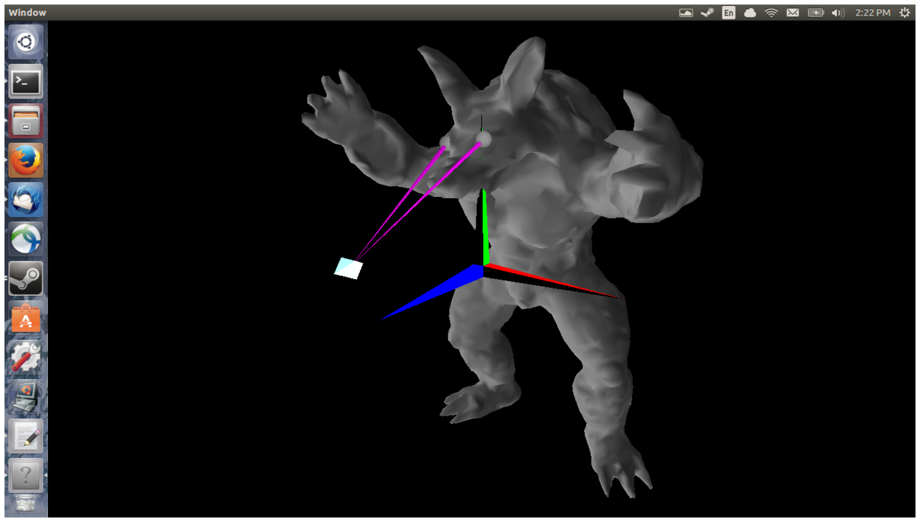

Figure 1: Eyes and Lasers pointed at the gem

(25 pts) Eyes. Create a new shader that displays the `g_eye` mesh. It should scale the mesh to an appropriate size, move it to the eye sockets, and rotate to point *at the gem*. As the user moves the gem around, the eyes should rotate to track it. Note that the untransformed eye-shape has its pupil pointing in the direction (0, 1, 0). You can use the same shader for each eye, but just pass in a different mat4 matrix from `main.cpp` that moves, scales, and rotates the eye to the proper configuration. For inspiration on how to edit `main.cpp` to load the new geometry, look at how it works for the armadillo. For your reference in building the matrices, the left eye socket is located, approximately, at world coordinates (-0.26454, 0.81828, 0.00855), and the right eye socket is at (0.04736, 0.86397, 0.05046). A possible scaling is 0.05.

(20 pts) Lasers. Create a new shader that displays a laser beam centred in an eye, shooting towards the gem (see Figure). Do this for both eyes. We recommend recycling the geometry of one of the coordinate axes, and just changing its colour to pink or red or whatever colour laser seems fitting for your armadillo. You'll need to transform the laser in the same way as you did the eyes, so that it is located and oriented properly. Then you'll need to scale it so that the base is small (we recommend .25) and elongate it so that the tip is located at the gem's position.

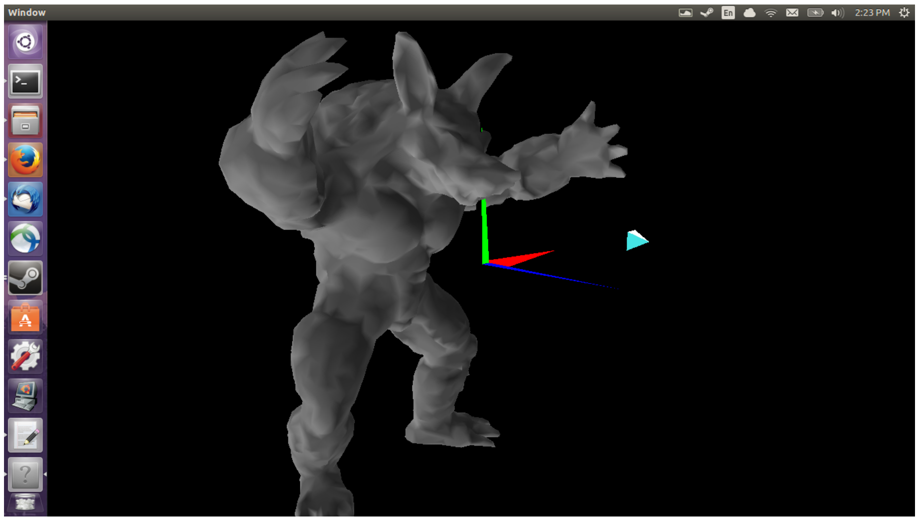

Figure 2: Rest position

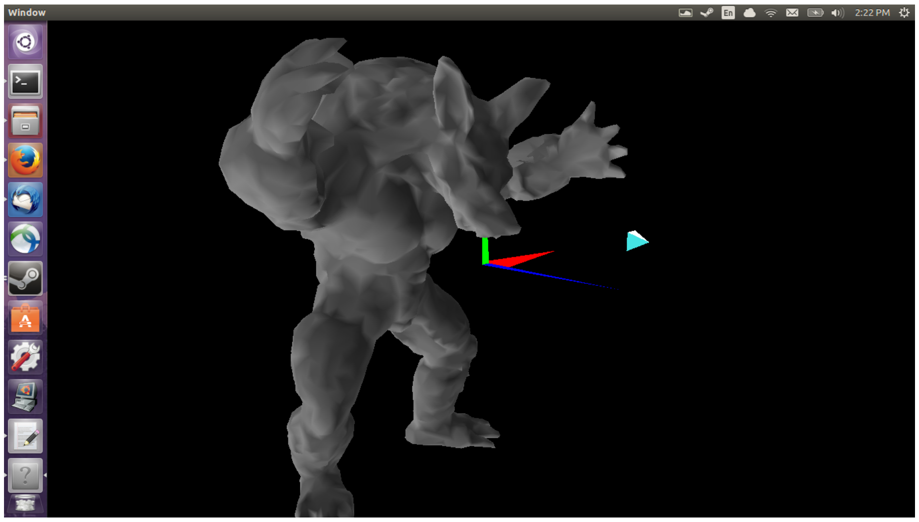

Figure 3: Rotated position

(25 pts) Head motion. For this part, we are simply rotating the head in a nodding motion. This is a simple rotation in only one dimension (about the pivot's x-axis). We will have two coordinate frames, which we will call the Neck frame and the Head frame. They will initially be perfectly aligned, located in the "neck" (we are aware that the Armadillo doesn't have much of a neck, but as an approximation you can locate the origin of the Neck around (0.0, 1.0, -0.3), with axes parallel to the world frame). The Neck frame is fixed to the body, while the Head frame is fixed to the Armadillo's head and moves with it.

Your first task is to determine, in the vertex shader, if the current vertex is in the head. One way to do this is to check whether vertex is inside an axis-aligned box defined in the Neck frame. For example, vertex position should have positive z coordinate. Pick the other sides of the box to fit the head.

Then, as the user presses the rotate keys, the Head frame should rotate relative to the Neck frame (the current rotation angle is stored in `head_rot_X`). Construct the appropriate Head matrix from the rotation angle (e.g., see `glm::rotate`), and pass it to the vertex shader. In the shader, transform the vertex position using this matrix.

Don't panic if some significant artifacts pop up around the boundary between the head and neck for larger rotations. This is a very simple modification, and there are some ways to improve it (see next part).

(30 pts) Creative License For this part we want to see what you can do. Your ideas should use at least one new shader, and should be of a similar complexity to the previous tasks. If you have any doubts, make sure to OK it with a prof or TA. Some suggestions:

- Add rotations to arms, or knees, etc. See Sec. 5.4 of the textbook.
- Linear blend skinning (also known as Skeletal Subspace Deformation) is a simple way to reduce, but not eliminate, the artifacts when parts of a mesh are deformed. This is a common technique used to animate characters in video games and other real time applications. The idea is to define a vertex's position as a linear combination of its position before head rotation and its position after rotation, that you computed in the previous part. To make this work, the key is to pick the weights so that the blending is limited to a small region around the neck, and change continuously from one end of the region to the other.
- Dance, dance, Armadillo! Dance!

Bonus marks may be given at the discretion of the marker for particularly noteworthy explorations.

Hand-in Instructions: You do not have to hand in any printed code. Create a README.txt file that includes your name, student number, and login ID, and any information you would like to pass on to the marker. Create a folder called "assn2" under your "cs314" directory. Put all the source files, your makefile, and your README.txt file for each part in the folder. Do not use further sub-directories. The assignment should be handed in with the exact command:

```
handin cs314 assn2
```