CS 175: Assignment 3
3D Coordinate Systems

Due: Monday, October 3th, 11:59pm

Assignment Objectives

Your program will use OpenGL to draw two cubes as well as some scenery, the cubes will become robots in the future. You will implement a number of different viewpoints, and allow the cubes to be moved around the scene. They will move with respect to various frames, including the two cubes' frames and external points of view. We provide skeleton code. You can download it off the git repo.

Step 1: Data Structures and Initial Setup (10%)

We provide you a matrix library which will allow you to compose affine transformations, apply them to vectors, and convert matrices to and from OpenGL's required format. You will use this matrix library to represent the coordinate transformations and frames for various objects. Recall, that for any object, we can express its frame, $\vec{\mathbf{o}}^t$, as $\vec{\mathbf{o}}^t = \vec{\mathbf{w}}^t O$ where $\vec{\mathbf{w}}^t$ is the world frame and O is some appropriate matrix.

Start with a single cube (or other simple object) in the center of the world frame. The cube should have a corresponding matrix that stores its frame's information.

Add in a second cube and set the matrices (describing rotation and translation for the cubes) such that the two cubes do not overlap. Also make these cubes different colors to easily distinguish them from each other.

Step 2: Complete `linFact` and `transFact` (20%)

Implement the functions `linFact` and `transFact` in `matrix4.h`. The purpose of these functions were described in class (also in Chapter 6 of the book).

Step 3: Views (10%)

Your rendering is done with respect to some eye frame. There is currently some default eye that has been set up. We want to now make the choice of eye frame interactive. Program in the ability to cycle between 3 different choices for the eye when the 'v' key is pressed:

1. The sky camera frame
2. The frame of cube 1
3. The frame of cube 2

Each time the 'v' key is pressed, the program should output which view mode it is currently on. (Recall our convention that cameras look down the negative z axis).

Step 4: Object Manipulation (60%)

For this part, getting the basic functionalities done (i.e., all modes of manipulation work and are usable) will get you 50% of the total grade. If you manage to make the mouse control match the spec exactly (i.e., you get the the sign of rotation angles and translation correct), you will get the remaing 10%.

Manipulation Modes

Now that we want to allow for greater user interaction, we'll have to track two key pieces of information (you may store them as application globals): (1) the object we wish to manipulate (2) $\vec{\mathbf{a}}^t$: the frame with respect to which it is manipulated. Make it so that pressing the 'o' key cycles through the different objects that we can manipulate. These objects are:

1. The sky camera
2. Cube 1 (later to be robot 1)
3. Cube 2 (later to be robot 2)

Depending on the current eye and current object being manipulated, we have a couple possibilities for $\vec{\mathbf{a}}^t$.

If the current object being modified is a cube and the current eye is the sky camera, then the only available frame for $\vec{\mathbf{a}}^t$ should be the *cube-sky* frame. This is a hybrid frame having the center coinciding with the cube's center, but with axes that are parallel to the sky camera frame's axes.

If the current eye is a cube, then all object manipulation is undefined (you may allow manipulation if you like, but you don't have to). There is one exception: you should not allow modifying the sky camera when the current camera view is a cube view.

If the current object being modified is the sky camera and the eye is the sky camera, pressing 'm' should switch between two frames for $\vec{\mathbf{a}}^t$:

1. *World-sky* frame: center at world's origin and axes aligned with sky camera.
2. *Sky-sky* frame: the sky camera's frame.

Mouse Movement

Now implement the actual response to user interaction. Allow the user to use the mouse to rotate and translate the sky camera and cube objects. The controls should be:

1. Left button: moving the mouse right/left rotates around the y-direction.
2. Left button: moving the mouse up/down rotates around the x-direction.
3. Right button: moving the mouse right/left translates in the x-direction.
4. Right button: moving the mouse up/down translates in the y-direction.
5. Middle button (or both left and right buttons down): moving the mouse up/down translates in the z-direction. For apple mouse, you need to press "Option/Alt" key + left button.

The signs of the rotations/translations should be inverted using the provided matrix inverse function if the current object being manipulated is the eye (this implies that the current view is the eye too). When in doubt, compare your signs to the solution executable.