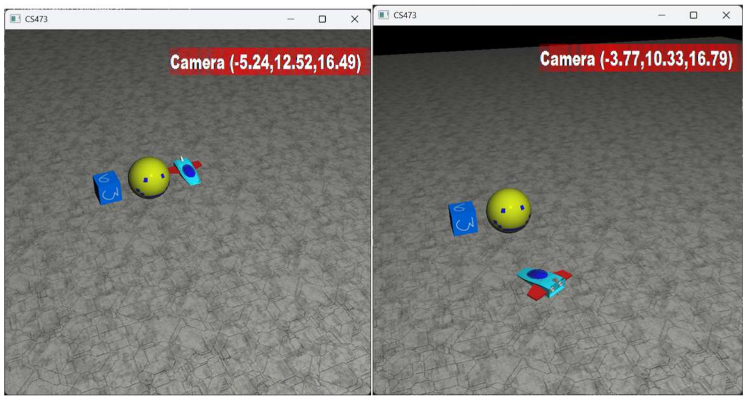
ICE 20.1: Make a Class to Manage an Imported Model

This tutorial assumes you successfully imported Blender Objects using the Lesson 19 starter code. This ICE helps you set up an Avatar class to permit objects that maintain and use their own location, orientation, and size within the world (and anything else you want to add).



1. Review classes ([C++ OOP (Object-Oriented Programming) (w3schools.com)]).
2. Create a new file, called avatar.hpp, with the usual header guards (ifndef/define/endif).
    a. Define a new class, called Avatar.
    b. Create the following protected data members:
        i. A BasicShape to hold the body of the ship (note you could add more BasicShape objects for more complex models).
        ii. A float for the *initial* rotation about the y axis (in degrees) to account for the orientation of the ship (likely needs to be rotated 90.0 degrees to align with the x-axis).
        iii. A float for the current rotation about the y axis.
        iv. A float for speed (per second).
        v. A glm::vec3 for the position of the object.
        vi. A glm::vec3 for scale.
    c. Create the following public member functions (declare in avatar.hpp and define in avatar.cpp—step 3):
        i. A constructor that accepts a BasicShape object, a float value for initial rotation about the y axis (default is 0.0), and a glm::vec3 for position. The constructor should set the current rotation to 0.0 and the scale to (1,1,1).
        ii. A destructor that deallocates the BasicShape objects in the avatar.
        iii. A member function called ProcessInput that accepts a GLFWwindow* and a float value for time passed as input and checks for keys to move the object. To start, consider the following movements.
            1. If forward (or UP) is pressed, move the avatar along the current azimuth (e.g., if the rotation is 60 degrees about the y axis, the position changes by (time_passed*speed*cos(radians(60)), 0.0, -time_passed *speed* sin(radians(60))) because the z axis is flipped.
            2. If left AND up are pressed, add a small amount (0.1) from the rotation angle.

        3. If right AND up are pressed, subtract a small amount (0.1) from the rotation.

    iv. A member function, called Scale, that accepts a glm::vec3 representing the scale of the object (e.g., (0.5,0.5,0.5) to reduce it to half its size).

    v. A member function, called Draw that takes a pointer to a shader program and draws the shape by first scaling it to the right size, rotate it using the orientation angle, rotate it again using the current rotation angle, then translate it by the object's position.

3. Create a new file, called avatar.cpp and define the member functions listed in 2.c.
4. Include avatar.hpp in main.cpp. Create a new Avatar object using the ImportOBJ::loadFiles to create the BasicShape object the avatar uses. Use the Avatar::ProcessInput and Avatar::Draw member functions to move and draw the new Avatar object.
5. Troubleshoot until the avatar moves like you like.