CSCE 441 - Computer Graphics

Programming Assignment 2

Deadline: Oct. 4th (11:59 pm)

1 Goal

The goal of this assigment is to become familar with model, view, and projection transformations in OpenGL.

2 Starter Code

The starter code can be downloaded from here.

3 Task 1

Download the code and run it. You should be able to see three cubes as shown in Fig. 1. Make sure you write your name in the appropriate place in the code, so it shows up at the top of the window. Here is a brief explanation of the starter code:

- There are two folders in the package. "shaders" contains the vertex and fragment shader programs. You do not need to touch these files for this assignment. We will discuss them later in the course. The other folder "src" contains the source files. Again, you do not need to touch the "Program.cpp" and "Program.h" as they are responsible for reading and compiling the shader programs. For this assigment, you'll be mainly modifying the "main.cpp" and "MatrixStack.cpp".

- The `MatrixStack` class is basically the matrix stack used for hierarchical transformation as discussed in the class. The stack is always initialized with the identity matrix. Any transformation is then right multiplies with the matrix at the top of the stack. `pushMatrix` creates a copy of the top and pushes it to the stack. `popMatrix` remove the top matrix from the stack. There are several transformation in this class that are currently implemented using the glm library. For more information about the library, access the API documentation. You will be implementing most of these transformations by commenting out the glm functions in the next task (Sec. 4). Moreover, you will be using this `MatrixStack` class to write the transformations required for implementing a functional robot as shown in Fig. 2.

- The `main` function in "main.cpp" is similar to the one in the previous assignment. The `Init` function, initializes the window, events, and the shader programs. It also calls the function `CreateCube` which creates an array of vertices and their colors representing a cube. The vertex position and colors are then passed to the GPU in this function to be accessible later in the main display loop.

- There are a few callback functions for the mouse, cursor position, and keyboard which you need to fill to handle the input based on the instruction given in later tasks.

- The `Display` function is the one responsible for drawing the cubes on the screen. This function sets the transformations using the global matrix stack variable `modelViewProjectionMatrix`. Specifically, it first sets the top matrix to identity and creates a copy of the top. Then it sets up the perspective and view transforms by calling the `Perspective` and `LookAt` functions. The next chunk of code is responsible for drawing the first cube. We first create a copy of the top matrix by calling `pushMatrix`. Then we perform a series of model transformation including translation, rotation, and scale to position the cube properly. Finally, we draw the cube by calling the `DrawCube` function and pop the matrix. This will only remove the model transformation for cube 1 while keeping the projection and view transforms in the stack. We continue the same process for cube 2 and 3. Note that, for every push there needs to be a pop.

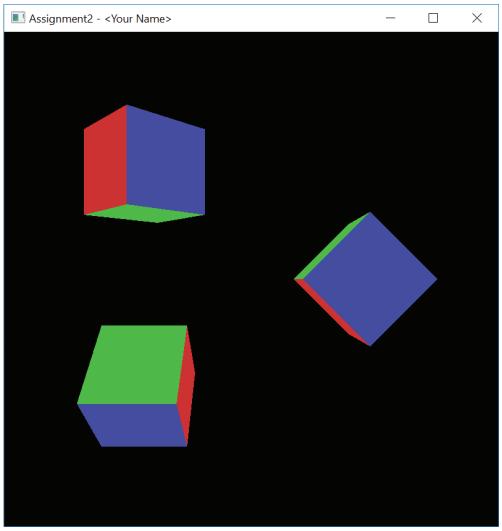

Figure 1: Running the skeleton code should produce three cubes as shown here.

- `ConstructRobot` function and `RobotElements` class are discussed later in Sec. 5.

4 Task 2

In this part, you will be implementing several 3D transformations that are currently implemented using the glm library in the `MatrixStack` class. Since each transformation is a 4×4 matrix, you can create a one dimensional array of size 16 (`float A[16]`) and fill it in based on the transformation appropriately. Note that glm is column major, so the indexing is as follows:

$$\begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix}$$

Once you fill in the array, you can feed it to the glm matrix using the `glm::make_mat4` function. Note that, you need to set all the values in your one dimensional array (even if the value is zero) since it is not initialized to zero when you create it (could contain garbage values).

Specifically, you need to implement the following functions:

- `translate(const glm::vec3 &t)`
- `scale(const glm::vec3 &s)`
- `rotateX(float angle)`
- `rotateY(float angle)`
- `rotateZ(float angle)`
- `multMatrix(glm::mat4 &matrix)`
- `LookAt(glm::vec3 eye, glm::vec3 center, glm::vec3 up)`
- `Perspective(float fovy, float aspect, float near, float far)`

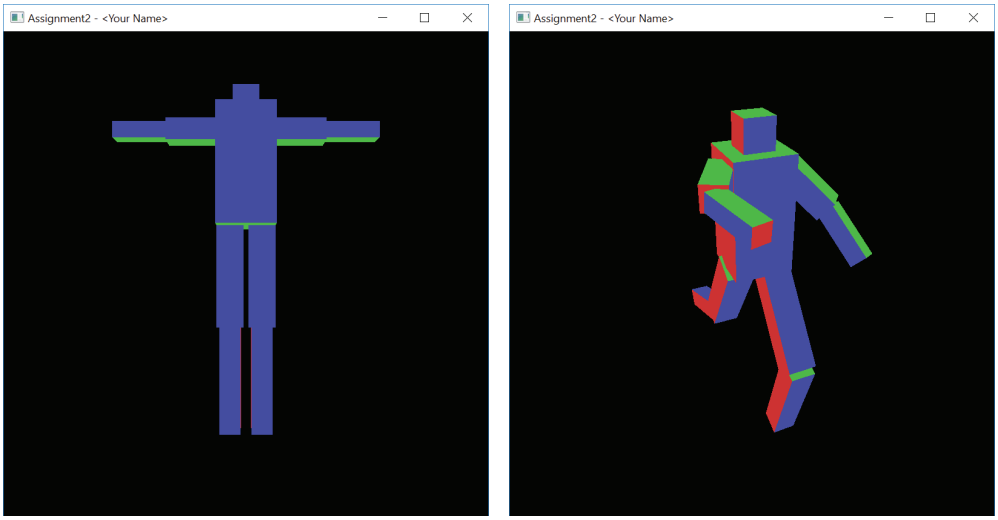

Figure 2: You need to create a robot with 10 components as shown on the left (the resting position). Different components should be rotatable in a hierarchical manner as shown on the right (i.e., rotating the upper arm, rotates the lower arm as well).

Note that, the function `multMatrix` is not a transformation, but right multiplies the input argument (`matrix`) with the top matrix in the stack. So for this, you need to implement matrix multiplication.

If you implement all these functions correctly, you should get the same three cubes, in the same position, scale, and orientation.

5 Task 3

You write a program to create a robot with 10 components (see Fig. 2) in a hierarchical way as follows:

- Torso
  - Upper left arm
    - Lower left arm
  - Upper right arm
    - Lower right arm
  - Upper left leg
    - Lower left leg
  - Upper right leg
    - Lower right leg

When a parent component is transformed, all of its descendants should be transformed appropriately. You should be able to transform each component using keyboard as follows:

- “.” (period): traverse the hierarchy forward
- “,” (comma): traverse the hierarchy backward
- “x”, “X”: increment/decrement x angle
- “y”, “Y”: increment/decrement y angle

- "z", "Z": increment/decrement z angle

By pressing the period and comma keys, you should be able to select different components in the hierarchy. You must draw the selected component slightly bigger by scaling it, so that it is distinguishable from unselected components. The x/X, y/Y, and z/Z keys should change the rotation angle of the selected component.

[Extra]: Animate a running/walking/etc. model by bending some or all of the joints with time, using `glfwGetTime()`. Animated joints do not need to be controlled with the keyboard.

5.1 Details

The class `RobotElements` represents a component of the robot. This class should contain the necessary member variables so that you can make a tree data structure out of these components. The tree should be constructed in `ConstructRobot` function and it should be called in the `Init` function. The root of the tree should represent the torso, which means that transforming the torso transforms everything else.

In addition to the member variables required for the tree hierarchy, the class should also have the following:

- A `glm::vec3` representing the translation of the component's joint with respect to the parent component's joint.
- A `glm::vec3` representing the current joint angles about the X, Y, and Z axes of the component's joint. (You may want to start with Z-rotations only.)
- A `glm::vec3` representing the translation of the component with respect to its joint.
- A `glm::vec3` representing the X, Y, and Z scaling factors for the component.
- A member method for drawing itself and its children.
- Any additional variable(s) and method(s) you see fit.

The drawing code should be recursive. In other words, in the `Display()` function in "main.cpp", there should be a single draw call on the root component, and all the other components should be drawn recursively from the root. The drawing function should take `modelViewProjectionMatrix` and update it by the transformation of each component (based on the position, angle of the joints, etc.) in a recursive manner. Make sure to pass the matrix stack by reference or as a (smart) pointer.

For this assignment, the 3D rotation of the joint should be represented simply as a concatenation of three separate rotation matrices about the x-, y-, and z-axes: Rx * Ry * Rz. The position of the joint should not be at the center of the box. For example, the elbow joint should be positioned between the upper and lower arms.

You must draw the selected component slightly bigger than the other ones. This requires you to scale the selected component. The traversal of the tree with the period and comma keys should be in depth-first or breadth-first order. Do not hardcode this traversal order - your code should be set up so that it works with any tree.

6 Task 4

Here, you will write functions to allow the user to change the viewpoint using mouse. For this, you need to change the `eye`, `center`, and `up`, which are the inputs to the `LookAt()` function, based on the user input. The mouse inputs are explained below:

- Holding the left mouse botton and moving the mouse should rotate the camera around the center point. Moving the mouse horizontally should rotate the view left/right. Similarly, moving the mouse vertically should rotate the view up/down.
- Holding the right mouse botton and moving the mouse shoud translate the camera. This means both the eye and the center should be moved with the same amound. Again moving the mouse horizontally and vertically corresponds to moving the camera left/right and up/down, respectively.
- Scrolling the mouse should change the distance of the camera to the center point. This means that the view direction stays the same, but the camera gets closer to or further away from the center. For this, you need to look up the appropriate callback function for scrolling from the GLFW mouse API documentation.

7 Deliverables

Please follow the instruction below or you may lose some points:

- You should include a README file that includes the parts that you were not able to implement, any extra part that you have implemented, or anything else that is notable.
- Your submission should contain folders "src" and "shaders" as well as "CMakeLists.txt" file. You should not include the "build" folder.
- Zip up the whole package and call it "Firstname_Lastname.zip". Note that the zip file should extract into a folder named "Firstname_Lastname". So you should first put your package in a folder called "Firstname_Lastname" and then zip it up.

8 Ruberic

Total credit: [150 points]

[30 points] - Implementing the `MatrixStack` class functions
  [10 points] - Implementing `translate`, `scale`, `rotateX`, `rotateY`, and `rotateZ`
  [05 points] - Implementing `multMatrix`
  [05 points] - Implementing `Perspective`
  [10 points] - Implementing `LookAt`
[75 points] - Implementing the robot with 10 components
  [50 points] - Implementing the hierarchical robot with recursive design
  [10 points] - Ability to control the robot with keyboard
  [15 points] - Ability to select components with the keyboard and show them with a different size
[45 points] - Changing the view
  [25 points] - Rotating the camera around the center
  [10 points] - Translating the camera
  [10 points] - Changing the distance of the camera to the center

Extra credit: [5 points]

[05 points] - Animating the robot

9 Acknowlegement

The robot part is based on the assignment by Shinjiro Sueda.