Assignment 2 Part B

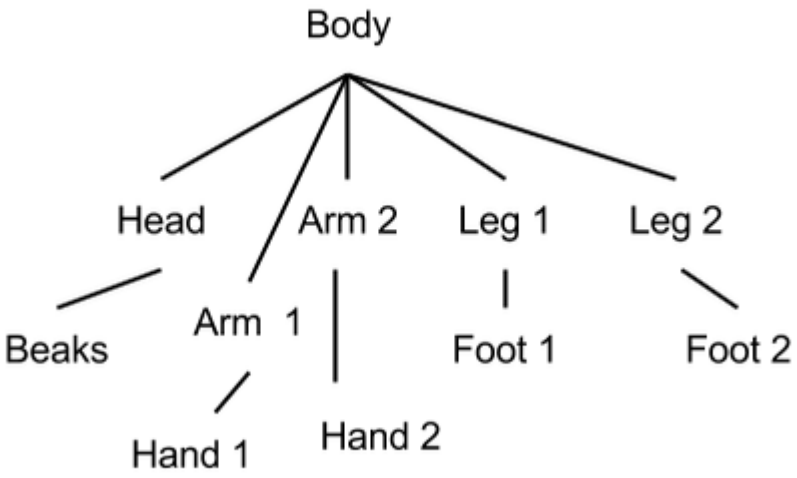
Report

Tanzim Mokammel
9966155208

I started by exploring the helper code. To grasp an understanding of the flow of the code, I started playing around with the given cube. The first step was to set a plan of action. I decided I would complete the code in the following order: designing the three basic rendering models, designing the penguin, designing the lighting system, ensuring the animation functionality worked by implementing the "update keyframe" button. The final step was to create a set of sample keyframes, to demonstrate the functionality of all of the components.

I started by trying the different rendering models on the cube. The solid functionality was already provided. Drawing the wireframe was simply a matter of using GL_LINE instead of GL_FILL. AS for the solid with the outline functionality, I had to draw every polygon twice. I implemented an if statement inside the polygon functions to draw with either LINE_LOOP, or GL_QUADS (or GL_TRIANGLE) depending on the argument from the main function. To avoid z-fighting, I had to enable the depth buffer, and use a polygon offset of (0.5, 2), which allowed the lines z-value to be offset, and therefore show up consistently.

Once the cube was able to render with the 3 models, I started drawing the penguin. This was the most time consuming component. I started off with the kinematic tree, which is as below:



Using the kinematic tree, correspoding push and pop mechanism were employed to allow for rotation and translation without issues. I am using knee and foot synonymously. Same applies for hand and elbow.

The procedure drawing for each component is as follows: draw the elementary polygon, color the polygon, draw the outline if requested, translate the center of rotation, scale proportionately, rotate for proper placement, and finally translate for placement. After these steps, rotation and translation based on the joint UI updates were setup, in that order. The body, head, and hands consist of trapezoidal prisms (front and back faces are trapezoid). The feet is made from triangular prisms. The rest of the body parts were made from the given unit cube. After each body part was made, the joint UI data was changed using the UI, and the proper rotational joint placement, as well as the rotations were verified.

Once the penguin was fully operable. I designed the lighting system. This required specifying the normal to each face of the elementary polygons. Normals for the rectangular prisms were easily determined. For the triangular and trapezoidal faces, I calculated the cross product manually on paper, using 2 of its vertex vectors. I used enable GL_LIGHTING, and GL_LIGHT0 for this functionality. The light control was then designed by modifying the location of the light source. It was placed at z=0, and moveable in a circle around the center, parallel to the x-y place, and infinitely far away. The light can be controlled in the Joint Control Window, within its own panel. This required a modification to keyframe.h. The lighting system was tested initially on the colored polygons (GL_COLOR_MATERIAL).

Once the light was verified functional, material properties were given to the penguin, and the matte and metallic entries were placed in the UI. After setting up the corresponding links between the ui and the display function, material properties we assigned. For proper lighting, ambient, diffuse, and specular components needed to be considered. OpenGL also allowed for an extra shininess variable. Using these factors, matte, and metallic were differentiated. The matte rendering contains an ambient and diffuse component. The metallic rendering contains an ambient, and specular component. The shininess is much lower on the matte rendering as well. These are the key differences in the material properties of the two rendering styles, to achieve the matte, and metallic look. For both cases, the polygon colors were disabled to allow full control on GL_MATERIAL properties.

To make for easier visibility, I added a few extra features. By modifying the mouse function, I can now rotate the mouse in x or y using the LMB. RMB can be fragged in y to change the light source angle. I also placed a small cube in a circle to represent the light source location, although the source itself is infinitely far away.

The last step was to code the update keyframe button. The loadKeyframesFromFIleButton was helpful. This process was simply understanding the classes in Keyframe.h, and utilizing the correct getter and setter functions. The enumerations were extremely helpful in appropriately updating the DOF vector. Rest of the animation functionality were already designed. Once everything was complete, I designed an animation with the following steps:

- Pick starting position at keyframe 0, and time 0
- Update time and keyframe, and use “Update Keyframe” after selecting a pose using Joint UI to modify DOF vectors

- Once keyframes have been designed, save the keyframes to file for future loading, rename the file avoid accidentally overwriting it
- Play back the animation

Sample keyframe files have been provided in a separate folder. Please rename to keyframes.txt prior to loading to animation. Below sample keyframes were submitted:

- A dance animation to demonstrate all joint movements
- A lighting demo, where the light source travels in a circle around the origin
- A flying animation