SJSU - CS 116A

Project 3 – Build a Hierarchical Bipedal Skeleton

Introduction

In this Project, you are going to use the knowledge you have gained in transformations to build a fully animatable Bipedal Skeleton . For the first part, we will create the tools and classes necessary to build the skeleton and write/read it to disk. For the next part we will create keyframe animation on the joints of the skeleton to animate the rig.

Part 1 Prerequisites

1. You will need to have a good understanding of the hierarchical transformations we learned in the class exercises and all the example code provided.
2. Know how to select drag and rotate objects with the mouse.

This is an individual assignment but please feel free post any questions/discussions to the class forum in the Discussion section of Canvas.

Part 1 – Implement the Skeleton Classes with Base Functionality

- First Step is you need to create a *Joint* class. You can choose to inherit from the Sphere class and override some of the it's functionality or you can inherit from *SceneObject* and write your own custom joint class from scratch. Your choice! The Joint class needs to support parenting to another Joint. Joints should also have a name (string). The name will be used for the file format (see below).
- Add the ability to create new joints and add them to your scene dynamically. Use a hot-key to do this (ex: "j" or "J") . If an existing joint is selected when you add a new joint, the selected joint should be come the parent of the new joint. The name of the joint can be assigned automatically as new joints are created. (ex: joint0, joint1, joint2…joint3). You can concatenate a number to the string name of the joint as new joints get created. Don't use the size of the stored array of joints to determine this number (why ?)
- Joints should be able to be selected, moved and rotated (about their center) with the mouse. Use the "x", "y", "z" keys as rotation modifiers to change the rotation value. It might be helpful to display text in your viewport, that shows the current x, y, z angle of rotation as you change it.
- Add a <delete> function to delete the selected joint and remove it from your scene. Make sure to reconnect orphaned child nodes to the graph.
- Write a custom Joint::draw() method that draws the joint (which can be a sphere), but also draws a segment (bone) to ithe parent (if the joint as a parent). The drawing style can be of your own design but it should be more than just a single line. (Suggest using a cone or a pyramid). *Note that the bone does not have to be selectable, only the joint.*
- Now that you have all the basic functionality to draw a skeleton using joints, create a saveToFile() function that can traverse your skeleton tree and save the structure to a script file. Use this file format file format to describe each node:

create -joint “joint name” -rotate “<x, y, z>” -translate “<x, y, z>” -parent “name”;

If none of the transformation options are present, the joint should simply be created with default values. (rotate (0,0,0), translate(0,0,0)). Note that translate and rotate values are relative to the parent.

If no parent is provided, then the joint is the root. You only need to support a single root and joint hierarchy for this version (ie one character), but if you want extend to support multiple roots, it’s not that difficult.

Example: *create -joint joint1 -rotate <45.0, 90.0, 30.0> -translate <2.0, 2.0, 2.0> - parent joint0;*

Note that “joint0” must exist first for this command.

Your program should be able to read the script you saved and create the skeleton.

You should be able to trade scripts with your classmates and recreate the same structure in your program.

- Create an example *bipedal skeleton* with your program with at least 24 joints. (note: if you want to create a quadruped, that is OK also,, but only a biped is required).
- Create a video showing your workflow (abbreviated), the completed skeleton and saving/restoring the file. Make sure to show testing moving, rotating joints.

Important - What to Submit

1. Submit only ofApp.h and ofApp.cpp files for your project (and any other sources files required to build your project). Zip it into a .zip file using the following naming convention:

   Project3-Skeleton1-<your name>-<date>.zip.

As an example:

Project3-Skeleton1-KevinSmith-09042018.zip

No other compressed format will be accepted (zip is available natively on both windows and Mac).

2. Submit a video showing your skeleton building workflow, manipulation (translate AND rotation) of the joints and your file save/restore capability.

Grading Criteria

To receive the highest grade on the assignment, the code most be robust and well documented (commented) with all functionality specified included. Source code files must be signed by the author.