CMPUT 411/511—Computer Graphics

Assignment 2

Fall 2017
Department of Computing Science
University of Alberta

Due: 23:59:59 *Sunday, October 29*
Worth: 20% of final grade for CMPUT 411
(15% of final grade for CMPUT 511)
Instructor: Dale Schuurmans, Ath409, daes@ualberta.ca

In this assignment you will implement a simple motion viewer (in C++/OpenGL) that can read in a description of a 3D skeleton with an associated motion capture sequence, display the skeleton in its initial pose, animate the motion sequence, and orient and move the camera. Also, you will add interpolation capabilities to the animation.

Note When submitting your program, you need to include a `Makefile` that allows the TA to simply run "`make`" and have your program compile properly on the lab machines. The TA will then run the executable "`./motionViewer` ⟨filename.*bvh*⟩" to test your program on the model stored in ⟨filename.bvh⟩.

When finished, please submit a *single* .zip archive containing your files on **eclass**.

Note A set of .bvh files containing examples of human motion capture sequences is posted on the assignment web page in a2files.zip. (A few thousand more can be downloaded from https://sites.google.com/a/cgspeed.com/cgspeed/motion-capture/cmu-bvh-conversion for your viewing pleasure.)

Implementing a viewer for motion capture sequences

You will implement a basic motion capture viewer, with executable called `motionViewer`, that reads in a Biovision .bvh file describing a simplified stick figure skeleton represented by a hierarchy of line segments (limbs) connected by rotational joints. The skeleton as a whole can be translated while each joint can be rotated independently. Note that each joint has three rotational degrees of freedom, which is represented in a .bvh file by Euler angles. In particular, a .bvh file will first specify the skeleton and then provide a sequence of global translations and Euler angle configurations for each joint in the skeleton.

Marks are given for the functionalities achieved by your viewer.

1. (4%) **Read in a hierarchical model and motion sequence from a .bvh file**

   Your program needs to accept a single argument that names an input file to read. In particular, your program will be invoked with a command "`./motionViewer` ⟨filename.bvh⟩", where ⟨filename.bvh⟩ is the name of the file to be read.

   The input file ⟨filename.bvh⟩ is assumed to be in .bvh format, described in the file bvh_file_specification.html in a2files.zip.

   Note Because the models are hierarchical, they are most naturally stored as trees, and almost all processing is naturally done by a recursive depth-first traversal through the tree structure.

2. (1%) **Write a hierarchical model and motion sequence out to a .bvh file**

   It is very useful to have a simple output function for debugging and marking purposes. Therefore, when the user types the character 'w', your program should write to the output file `output.bvh`, recording the skeleton model, with all offsets, translation and angle specifications in the proper configuration, followed by the motion sequence data. In particular, your program should write out a valid .bvh file that represents the skeletal model and motion sequence that were read.

3. (2%) **Display the initial pose**

   Note The default initial pose for the figure in a .bvh file which has the figure standing upright in the $+y$ axis direction, facing forward in the $+z$ axis direction, with the left side facing toward the $+x$ axis direction.

   Use a perspective camera model. Define an initial view frustum that captures the entire figure in the camera's field of view (keeping the figure large enough to remain recognizable). Make sure that the camera-up direction coincides with the figure-up direction.

   Display the stick figure in the default initial pose, drawing the figure in white against a black background.

4. (5%) **Animate the sequence**

   When the user types:
   'p': play the animation, running in a continuous loop that wraps around at the end of the sequence back to the beginning (use a default animation speed of 120 frames per second),
   'P': pause the animation,
   'x': stop the animation, restore and redisplay the figure in its initial configuration,
   'q': exit the program.

   Note Use double buffering to ensure good animation quality.

5. (2%) **Place the camera in a good viewing position and orientation**

   If your camera placement in Question 3 is not already guaranteed to do so, define an initial view frustum that captures the entire figure in the camera's field of view *throughout the entire motion sequence*, keeping the figure large enough to remain easily recognizable. This will require you to do some simple pre-analysis of the skeleton and the sequence of translations.

   Make sure that the camera-up direction coincides with the figure-up direction.

6. (3%) **Add camera control**

   When the user types:
   'd' or 'D': translate the camera -1.0 or 1.0 units in the camera x direction (dolly),
   'c' or 'C': translate the camera -1.0 or 1.0 units in the camera y direction (crane),
   'z' or 'Z': translate the camera -1.0 or 1.0 units in the camera z direction (zoom),
   't' or 'T': rotate the camera -10 or 10 degrees counterclockwise around the camera x axis (Tilt),
   'a' or 'A': rotate the camera -10 or 10 degrees counterclockwise around the camera y axis (pAn),
   'l' or 'L': rotate the camera -10 or 10 degrees counterclockwise around the camera z axis (rolL).

   Note Keep the animation running at the same speed after each camera control keystroke.

7. (3%) **Control animation speed by interpolating the motion sequence**

   When the user types:
   '-' or '+': decrease or increase the virtual speed of the animation by 10 frames per second respectively (allowing the virtual speed to become negative, which means *reversing* the animation),
   'x': stop the animation, *reset the virtual speed of animation to the default*, restore and redisplay the figure in its initial configuration (i.e. just modify the previous stop function to reset the animation speed).

   Note Adjusting the virtual animation speed means still displaying 120 poses per second, but changing the number of original poses that are spanned in a second, by using interpolated poses. For example, if '-' has been hit four times, then only 80 of the original poses should span a second, meaning that for every two original poses you need to instead display three interpolated poses.

   Note Use simple linear interpolation for translations, and spherical linear interpolation for all rotation angles. (Remember that linearly interpolating Euler angles does not work! Instead you can use quaternions to linearly interpolate angles properly.)

Note Overall, it is a good idea to debug your animations by first using only simple fake motion sequences. (You can just replace the motion in any .bvh file with a simpler debugging motion sequence.) For example, start with only having translations of the entire figure; no rotations of any joints. Then try rotating only one joint. Then try rotating only two connected joints. Once these work properly separately, try them together. Once that works, your code is probably okay and you can start to try it on some of the real motion capture sequences.